1-1-2020

Robust and Efficient Activity Recognition from Videos

Xin Li
Lehigh University

Robust and Efficient Activity Recognition from Videos

by

Xin Li

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Computer Science

Lehigh University

(January 2020)

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____
Date

_____
Professor Mooi Choo Chuah, Dissertation Advisor

_____
Accepted Date

Committee Members:

_____
Professor Mooi Choo Chuah, Committee Chair

_____
Professor Brian D. Davison

_____
Professor Jeffrey D. Heflin

_____
Professor Wei-Min Huang
(Math Department, Lehigh University)

_____
Professor Ting Wang
(Pennsylvania State University)

To my family.

Acknowledgements

I would first like to express my gratitude to my advisor, Prof. Mooi Choo Chuah, for her continuous support during my entire doctoral period at Lehigh University. Prof. Chuah has been an excellent mentor. She taught me how to organize a paper clearly and present my work professionally. More importantly, she trained me to become a good researcher. I learned how to search and read papers efficiently, how to think critically, and how to propose ideas creatively. Thus, it is my great honor to study for a Ph.D. degree under the supervision of Prof. Chuah.

I am grateful to all of my committee members Prof. Brian D. Davison, Prof. Jeffrey D. Heflin, Professor Ting Wang, and Prof. Wei-min Huang. Their constructive comments on my thesis and valuable suggestions on my oral defense significantly improve the presentation of this thesis. I want to acknowledge their time and energy.

I also want to thank Dr. Ning Bi, Dr. Yingyong Qi, Dr. Bolan Jiang, Dr. Shuai Zhang from Qualcomm, Dr. Hongxia Jin, Dr. Dawei Li from Samsung for guiding and helping me when I was doing summer internships. The projects I did at Qualcomm and Samsung have become parts of this thesis. I would like to thank my collaborators during my summer internships. What I learned from you helps enhance the quality of this thesis.

Many thanks to my labmates from WiNs lab: Dawei Li, Qinghan Xue, Wenbo Li, Zhiyu Chen, and Xiaowen Ying. I enjoy our friendships and the fun activities we had together. I appreciate so many inspiring discussions with you. Besides, I want to thank my friends Hongyang Zhang, Peng Gao, Rui Cheng, Shiyu Zhang, Kexin Chen, and Yang Yi. You have been with me all the time.

Finally, I would like to extend my gratefulness to my family especially my parents for their emotional and financial support over the years. I am most grateful for my fiancée, Yuan Qi, for her love, understanding, and companionship.

Contents

List of Tables

List of Figures

Abstract

With technological advancement in embedded system design, powerful cameras have been embedded within smart phones, and wireless cameras can be easily deployed at street corners, traffic lights, big stadiums, train stations, etc. Besides, the growth of online media, surveillance, and mobile cameras have resulted in an explosion of videos being uploaded to social media sites such as Facebook and YouTube. The availability of such a vast volume of videos has attracted the computer vision community to conduct much research on human activity recognition since people are arguably the most interesting subjects of such videos. Automatic human activity recognition allows engineers and computer scientists to design smarter surveillance systems, semantically aware video indexes and also more natural human-computer interfaces. Despite the explosion of video data, the ability to automatically recognize and understand human activities is still rather limited. This is primarily due to multiple challenges inherent to the recognition task, namely large variability in human execution styles, the complexity of the visual stimuli in terms of camera motion, background clutter, viewpoint changes, etc., and the number of activities that can be recognized. In addition, the ability to predict future actions of objects based on past observed video frames is very useful. Therefore, in this thesis, we explore four designs to solve the problems we discussed earlier, namely (1) **A semantics-based deep learning model, namely SBGAR, is proposed to do group activity recognition.** This model achieves higher accuracy and efficiency than existing group activity recognition methods. (2) Despite its high accuracy, SBGAR has some limitations, namely (i) it requires a large dataset with caption information, (ii) activity recognition model is independent of the caption generation model and hence SBGAR may not perform well in some cases. To remove such limitations, **we design ReHAR, a robust and efficient human activity recognition scheme**. ReHAR can be used to recognize both single person activities and group activities. (3)

In many application scenarios, merely knowing what the moving agents are doing is not sufficient. It also requires predictions of future trajectories of moving agents. Thus, **we propose GRIP, a graph-based interaction-aware motion intent prediction scheme**. The scheme uses a graph to represent the relationships between two objects, e.g., human joints or traffic agents, and predict the motion intents of all observed objects simultaneously. (4) Action recognition and trajectory prediction schemes are typically deployed in resource-constrained devices. Thus, any technique that can accelerate the computation speed of our schemes is important. Hence, **we propose a novel deep learning model decomposition method called DAC** that is capable of factorizing an ordinary convolutional layer into two layers with much fewer parameters. DAC computes the corresponding weights for the newly generated layers directly from the weights of the original convolutional layer. Thus, no training (or fine-tuning) or any data is needed.

Chapter 1

Introduction

Recent technology advancements have made high-quality and affordable cameras available in many gadgets, e.g., smart-phones, wireless cameras, autonomous vehicles, that humans own these days. Analyzing images/videos captured by these cameras impacts our daily lives. For example, smart-phones have been using face recognition algorithms [118, 20, 16, 75, 140, 122] to analyze frames captured by front-cameras (RGB or infrared camera) to recognize users, that improves the security and usability of smart-phones. Smart surveillance video systems which can detect and identify suspects [42, 102, 32] help law enforcement personnel maintain a safer living environment. Hand gesture recognition algorithms [45, 131, 117, 188] provide a brand new way for human-computer interaction interfaces to be designed.

Apart from these examples, automatically recognizing (or predicting) human activities in videos also attracts the interests of researchers due to its broad application scenarios, such as healthcare systems [13, 70, 105], video search engines [83, 25, 29], sports analysis systems [189, 150, 64, 129, 143], surveillance video systems [5, 94, 95, 160, 84, 126], self-driving cars [152, 103, 96, 38, 156, 170], and so on. Human activity recognition (prediction) algorithms help an intelligent system gets a better understanding of the contents of a video. By monitoring and analyzing the behaviors of elderly or patients, a healthcare system is able to quickly notify medical doctors once it detects harmful behaviors, e.g., a fall, so that the elderly or patients can get timely treatment. Using human activity recognition, video search engines can search content similar (activity similar) videos, and sports analysis system can quickly locate and highlight several exciting short video

clips from a long video to help users save time by avoiding watching entire long videos. Self-driving cars equipped with human activity recognition and prediction schemes can liberate drivers from steering wheels so that they can do more meaningful things, e.g., read business news. Most of the computer vision related systems require high accuracy, e.g., the face authentication system for unlocking smartphones, self driving cars and healthcare systems that analyze patients' gaits to diagnose Parkinson diseases.

Other than accuracy, most of these computer vision related systems also require highly efficient computer vision algorithms. Even though more and more deep learning models have been deployed on mobile devices, some of such models require the phones to be connected to the internet since the deployed models are partly run on mobile devices and partly in the cloud due to the resource constraints of mobile devices. Human activity recognition and motion intent prediction models are great examples that are known to suffer from slow computation time. Users of intelligent computer vision applications can benefit from having efficient algorithms running in real-time on resource constrained devices. Similarly, self-driving cars only have limited computing resources and hence need efficient object detection and trajectory prediction algorithms.

In this thesis, we set out to address some of the design challenges in designing human activity recognition and motion prediction algorithms that are often used in smart computer vision systems. Our specific goals are to explore more efficient algorithms with high accuracy that can be deployed on resource-constrained devices. Specifically, we design two novel human activity recognition schemes that can recognize activity classes from videos, an efficient motion prediction method that is useful for human action prediction or motion planning in self-driving cars. Last but not least, we also propose a novel scheme to accelerate deep learning based models. For the remaining of this chapter, we will discuss the challenges of each task (human activity recognition, activity prediction and model accelaration) and review recent approaches for these tasks and their limitations.

1.1 Human Activity Recognition

A human activity recognition system takes a video clip (a sequence of video frames) as its input and classifies this video clip into an activity class label among several potential classes. Compared to generic AI tasks, e.g., object recognition from images [145, 146, 153, 62, 63], human activity

recognition is more challenging due to the complexity of human actions and the following factors:

- **Variation of videos:** Similar to images, different videos in a dataset may have varying lighting conditions, varying sizes and shapes of objects, partial or complete occlusion, cluttered background, crowded scenes, different viewpoints of the camera, etc.. Furthermore, considering a video consists of dozens of frames, continuous frames from the same video may also have changing lighting, moving viewpoint, varying sizes and shapes of objects, growing occlusions, shaking frames due to the motions of the cameras, and so on. Thus, it is extremely challenging to design a robust human activity recognition system that can perform well under all of these conditions.

- **Feature Representation:** Feature representation is always the key to the success of a machine learning model. Good feature representation is robust to the variations of data, highly discriminative for for various activity classes, and can generalize to different datasets. It is not easy to extract good feature representations from videos due to their high resolution and durations as well as the presence of noise due to environment factors and different video compression schemes. In addition, extracted features for human activity recognition should include spatial and temporal information simultaneously. The spatial information indicates the environment where the current activity happens and the apperance information of other objects, while the temporal information involves the motion clues of objects and any changes in the background. Moreover, designing a model that can extract features from video clips that can be used for recognizing both single-person activities and group activities is difficult, considering these two tasks require different levels of features. Group activity recognition requires higher-level semantic features than single-person activity recognition, e.g., how different persons interact with one another in a group activity.

- **Efficiency/Accuracy Tradeoff:** Processing high dimensional data, e.g., videos, usually requires more computing resources and longer processing time. As we mentioned above, good feature representation determines the recognition accuracy of a recognition system. Furthermore, a good feature extractor that can extract relevant spatial and temporal features simultaneously is often complex and runs slowly. If we seek a fast human activity recog-

nition system, it usually means that we cannot use complex structures for feature extractor and recognizer, which may result in accuracy degradation. Thus, it is challenging to design a model that runs fast while maintaining high recognition accuracy.

Related Work. In the past several years, many human activity recognition solutions have been proposed. Based on the type of extracted features, we can group these solutions into the following categories.

(1). Traditional Manually-designed Features: A few years ago, researchers manually designed features for human activity recognition. For example, Burkert et al. [11] fuse color and shape features, authors in papers [109, 142, 28] extract motion flow features, foreground extraction and region detection are used in [50, 17, 171], keypoint descriptor features, e.g., SIFT (Scale-invariant Feature Transform) and HOG (Histogram of Oriented Gradients), are employed in papers [187, 73, 30, 51]. Based on common experience, a model that only uses traditional manually-designed features runs fast. However, they are not robust to the variations of videos, and it is impossible for researchers to pre-design suitable features that can handle all possible environmental changes after they are deployed.

(2). Learnable Features: Recently, deep learning provides a way for researchers to automatically learn suitable features from some training samples. Convolutional Neural Network (CNN) models are proposed to learn frame-to-frame representation [6, 74, 8] and achieve much higher accuracy than only using traditional features. However, these CNN models merely learn an overall feature for the entire video without considering temporally-specific information. Then, Recurrent Neural Network (RNN) models are applied in [37, 64, 128] to fix the problem. Usually, researchers first use a CNN model to extract spatial visual features from each video frame and then feed these features to an RNN model to make a recognition. Such a structure gets better results than prior work. However, some researchers argue that CNN+RNN structure does not consider enough motion information. Thus, C3D model [71], I3D model [169], and 2+1D structure [158] are proposed for learning a spatio-temporal feature for human activity recognition. These models are trained to extract spatial and temporal features by either using 3D convolutional operations that collect spatial and temporal information simultaneously or alternately using 2D (on spatial) and 1D (on temporal) operations. Even though they achieve higher accuracy, they require a large amount of training

data and much longer training time. Thus, typically such solutions are not suitable for lower end products e.g. internet of things. Neither are such solutions feasible for academic researchers that do not have access to a large number of fast GPUs.

(3). Multi-stream Feature: Most of the models we discussed just now use a single feed-forward structure (single branch) to recognize human activities. Some other models consist of two or more streams (or branches) for better feature learning. In [144], Simonyan et al. first propose to use two-stream CNN networks structure for recognizing human actions in videos. Their model consists of two CNN models, namely Spatial stream ConvNet and Temporal stream ConvNet respectively, that share the same structure (five convolutional layers and two fully connected layers) but are trained for different weights. The Spatial stream ConvNet chooses one frame from the input sequence to extract spatial features, considering that some actions are strongly associated with particular objects and environment. The Temporal stream ConvNet takes multi-frame optical flow images (manually calculated using two continuous frames) as input to extract motion features. Then, the authors in papers [44], [164], and [14] improve the performance of this structure by either proposing a better feature fusion strategy or adding 3D convolutional operations at intermediate layers. Zhu et al. [190] apply a MotionNet to replace manually calculated optical flow image. Thus the entire model is trainable end-to-end, and the trained motion features are more specific and useful for those activities that appear in training data. After that, Han et al. [56] and Chen et al. [18] employ deeper and more complex networks to extract more robust features. These methods usually take manually trimmed video clips as input, which is not practical in real application scenarios. Some researchers use a fixed sliding window to crop clips, instead of manually trimming, from a video before feeding them into these methods. However, such a way has a high possibility to feed an unexpected activity, e.g., unseen background, into a model. Thus, the model will perform worse under such a situation.

1.2 Motion Intent Prediction

Motion intent prediction is an important function in many application scenarios, e.g., smart surveillance video system or self-driving cars. The system predicts objects' locations in the near future by observing their history trajectories. There are two big challenges when designing such a system.

- **Efficiency:** The prediction process should be faster than the activity itself. If the prediction process cannot finish before the activity finishes, then there is no need to use the system. Especially in the self-driving application scenario, the human motion intent prediction scheme will be run on an embedded system (with limited power and computation resources), computation efficiency is extremely important. Besides, the motion intent prediction scheme most likely needs to be applied to process videos that consist of multiple pedestrians, vehicles, or other objects, which means the schemes should predict motion intents for all of these objects simultaneously as fast as possible.

- **Accuracy:** The motion intent prediction scheme is designed for a system to respond before an activity happens. However, if a scheme makes a wrong prediction, then the system cannot respond correctly, which may result in a worse result than without using such a prediction scheme. Thus, high accuracy is required.

Related Work. Based on different types of inputs and outputs, we divide the motion prediction task into two categories. (1). Future-frame prediction. Some work [106, 107, 114, 119, 123, 138, 149, 173, 98] has been done to predict future frames. These solutions take previous one or multiple frames as input and generate frames in the future using some generative models, e.g., Generative adversarial network (GAN). Their purposes are merely focusing on predicting realistic pixel values in future frames. If we want to apply such solutions in self-driving cars or smart surveillance video systems, we still need other object detection and tracking schemes to understand the generated future frames. This pipeline cannot meet the efficiency requirement of our application scenarios. Thus, in this thesis, we are not going to discuss this direction (future-frame prediction) further. (2). Location coordinates prediction. Location coordinates prediction is based on the understanding of the current environment. For example, for human motion prediction, we assume the key-points, e.g., joints, of a human body have been detected and located, and the model is going to predict poses in the future; for traffic-agent intent prediction, the location coordinates of each object are expected to be collected, and the model will predict their future locations. Next, we will discuss existing schemes for human motion prediction and traffic-agent intent prediction separately.

For human motion prediction, Martinez et al. [113] propose a seq2seq recurrent neural network (RNN) model with a residual connection between the input and the output of each RNN cell. Such

a model can predict future motion for multiple actions, while its previous works only focused on building action-specific models. Then, in [21], Chiu et al. propose a new action-agnostic method for short and long-term human pose forecasting by modeling the hierarchical and multi-scale characteristics of the human dynamics. The biggest contribution of this paper [21] is that they model human dynamics in visual scenes by encoding the temporal dependencies of different time-scales in a hierarchical interconnected sequence of RNN cells. Besides, Pavllo et al. [124] propose a QuaterNet to represents rotations of each joint with quaternions, and its loss function performs forward kinematics on a skeleton to penalize absolute position errors instead of angle errors. With such an approach, they achieved better experimental results than existing works. Although all of these schemes perform better than existing solutions, they ignore the impacts of connected joints, which results in them suffering from missing structural information of human bodies. We argue that considering the structural information can achieve a better prediction result. The reason is that the motion of a joint is never independent, and it is impacted by other related joints. Thus, a model will make a wrong prediction, especially when making a long-term prediction, if it only considers one particular joint while ignoring the motion of other associated joints.

Motion prediction is also useful in other scenarios. For example, in self-driving vehicles, the vehicle needs to predict the movements of nearby objects in the near future, so that the system can proactively react toward these intents for safety purposes. As discussed in [54], most of the existing work on mobile agent motion prediction is either Markovian maneuver intention estimation-based [92] or prototype-trajectory based. Some researchers [155, 61, 137, 159, 136] proposed to predict future locations by recognizing maneuver (change lanes, brake, or keep going, etc.). However, these methods fail to predict the intents of objects accurately when they recognize the type of maneuver wrongly. The prototype-trajectories based approaches [91, 130, 46] are more robust to measurement noise when compared to the maneuver intention estimation-based approaches. However, the prototype-trajectory based approaches are typically computationally expensive and hence may be slow in detecting changes in pedestrian (or other objects) intent and are not suitable for self-driving scenarios. To enjoy the benefits of both, the authors in [19] use a combination of the two to develop a dictionary learning algorithm called augmented semi non-negative sparse coding (ASNSC). However, ASNSC predicts the intents only based on the spatial features while ignoring

the environmental context that may influence an object's intent. To handle this problem, Luo et al. proposed a convolutional network for fast object detection, tracking, and motion forecasting in [110]. Their model takes a series of bird's eye view LiDAR data as input and processes 3D convolutions across space and time, and then predicts the bounding box over the current frame as well as several frames in the future. They argue that such a structure is able to forecast motion because the model takes multiple frames as input and can learn velocity and acceleration features. However, the forecasting branch simply takes the 3D convolutional feature map as an input, so visual features of all objects are represented in the same feature map. In this case, the model will lose track of objects and hence cannot perform well in a scene that consists of crowded objects. Deo et al. [35] use convolutional social pooling layers to capture the interdependencies of the motion of all cars in the scene. Such a model indeed improves the accuracy of future motion prediction, because it has access to the motion states of surrounding objects and their spatial relationships. Although all of these models take the trajectory histories of all objects in the scene as their inputs, they merely predict the trajectory of one specific car (the one in the middle position) each time. Hence, these existing approaches require intensive computation power if they want to predict trajectories of all surrounding objects, which is highly inefficient, especially for autonomous driving cars scenarios.

1.3 Deep Learning Model Acceleration

Deep Learning Model acceleration aims to design a brand new structure or speed up a pre-trained model so that we can run complex deep learning models on resource-constrained devices in real-time while maintaining high accuracy. Designing such an acceleration solution, one will face the following challenges:

- **Tradeoff between Compress Ratio and Accuracy Drop:** Algorithm acceleration is known as a technique at the expense of accuracy. This statement also holds for accelerating a deep learning model. It basically means using fewer parameters when it comes to deep learning model acceleration. On the one hand, if we are designing a brand new structure with fewer parameters, it is highly possible that we cannot train such a model to achieve a satisfying accuracy. The reason is that a model with fewer parameters has an inadequate capability to

fit training data. On the other hand, if we are trying to speed up a pre-trained model, using fewer parameters definitely cannot reconstruct pre-learned functions, which will also cause accuracy drops. Thus, it is difficult to maintain high accuracy while accelerating a deep learning model.

- **Hardware Friendly:** When speeding up deep learning models, we want to run models on physical devices with a faster speed instead of only reduce the number of parameters. In a deep learning model, different structures have different running speed. For example, compared to a single feed-forward model, a model with skip connection or tensor reshape operations is usually slower. Even the kernel size of each layer matters a lot as discussed in [174]. Additionally, many factors of the running environment impact the speed, e.g., computing unit, memory size, operating system, basic function implementation, etc.. Thus, it is challenging to make sure an accelerated deep learning model is hardware-friendly.

Related Work. Many techniques to reduce the size of neural network models, e.g., model quantization of neural network models using fewer bits, have been proposed to facilitate their implementations on mobile chips [177, 175, 176, 172]. However, limited by current hardware structure and the tolerance for model accuracy drop, most of these quantization methods for real applications only focus on the 8-bit format. To further accelerate neural network models, it is more important to reduce computation complexity directly from the network architectures. Some research [62, 135, 184, 108, 94, 95] has been done to simplify these models before running them on mobile/IoT devices. Such research can be roughly categorized into two classes:

(1). Designing new light-weight network architectures: MobileNet proposed by Howard et al. in [62, 135] is an excellent example. The model is based on a streamlined architecture that uses depthwise separable convolutions to build a light weight deep neural network. The model achieves good accuracy and runs fast on mobile devices. Similar with MobileNet, ShuffleNet [184, 111] is another type of light weight network architecture, based on depthwise separable layers for acceleration. However, these models require powerful servers and massive data to tune the weights. This is not a friendly solution to those who cannot access such resources.

(2). Modifying an existing model to a slim version: Another solution is to produce a slimmer version of an existing model. Unfortunately, the training data in some cases is exclusively available

to the original designer of a model, which prevents other researchers from re-training the model after modification. Besides, it is costly and time-consuming to train a model from scratch. Thus, compared to designing new models and training them from scratch, accelerating an existing model based on its pretrained weights is a better solution. Network pruning and parameter decomposition are two common methods for this purpose. **Network pruning** is a practical tool for speeding up existing deep neural networks [116]. He et al. propose a channel pruning method [57] that utilizes LASSO regression to prune the number of the input channels in each convolutional layer. Even though such network pruning scheme simplifies models, it still has some weaknesses. Network pruning is based on the statistical results of a set of samples. Thus, (a) it still requires data to discover which channel to prune, and (b) the accuracy of the model drops after pruning because the statistical results are not suitable for all data during testing. **Parameter decomposition** is another way to simplify an existing model. It is a layer-wise operation that decomposes a layer into one or multiple smaller layers, either having smaller kernel sizes or fewer channels. Although there will be more layers after being decomposed, the total number of weights and the computational complexity will be reduced. For example, spatial decomposition [68], channel decomposition [185], CP decomposition [89], and Tucker decomposition [76], etc.. The decomposition methods only use the pre-trained weights of a layer, with the fact that most neural network models have much redundant parameters and can be largely simplified with low rank constraints.

1.4 Important Building Blocks

Before describing the details of our proposed schemes, we give a brief introduction to some building blocks. We mentioned these methods several times in earlier discussion and will discuss how we utilize these building blocks in our proposed solutions in subsequent chapters.

1. Optical Flow: Optical Flow was first proposed by Horn et al. [59]. It is used to describe how each point in the scene moves from a frame to the next. Many improvements have been introduced [10, 166]. Recently, machine learning methods [133, 93, 48, 66] have been used to estimate optical flow by taking two images as their inputs. Among all of these solutions, FlowNet 2.0 [66] achieved the most impressive results by using a stacked structure and fusion network.

2. Image Feature Extraction Via CNN: Convolutional Neural Network (CNN) [90] is a type

of feed-forward artificial neural network. It has been widely used in solving different types of tough tasks, e.g. natural language processing [9], image recognition [88], etc.

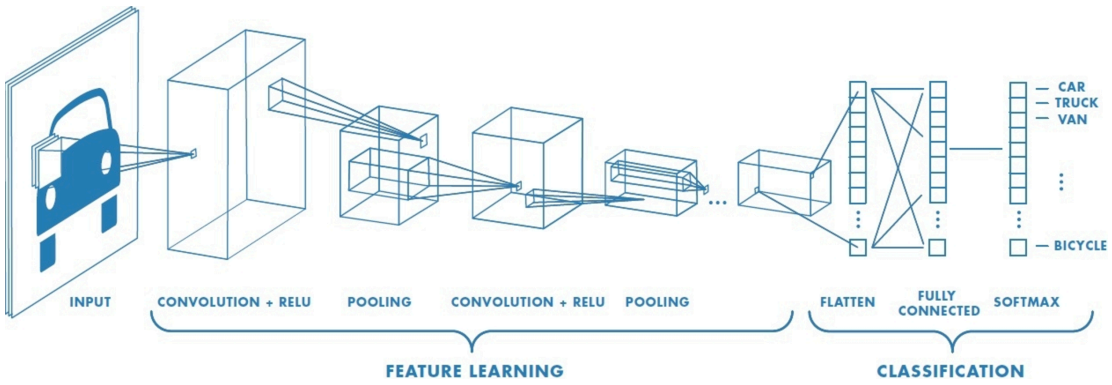


Figure 1.1: A shallow architecture example of a CNN. (Source[i])

As shown in Figure 1.1, a CNN usually consists of several convolutional operation layers, activation layers, and pooling layers. Each convolutional layer consists of a set of learnable filters. The filter slides across the input and computes dot product between the filter and the corresponding values of the input. Activation layers, e.g., ReLU, Sigmoid, Tanh, etc., are used to provide non-linearity to a CNN. The function of pooling layers is to reduce the number of parameters by reducing the spatial size of the representation progressively (the spatial size of the output is smaller and smaller as the depth of the CNN model goes deeper). Researchers usually use the output of the layer before fully connected layer (hidden layers or flatten layer in Figure 1.1) as an extracted CNN feature representation of the current input image. It has been proved that the CNN features contain more representative information of an image than other manually designed feature, e.g. SIFT, by Fischer et al. [47]. Furthermore, Donahue et al. [37] used CNN as a feature extractor to recognize human activities from videos, which showed that CNNs could extract useful information related to activities.

3. Long Short Term Memory: A Long Short Term Memory (LSTM) model is a particular type of Recurrent Neural Network (RNN) that was first proposed by Hochreiter et al. [58]. Because of its more powerful update equations and appealing back-propagation dynamics, the LSTM Network works slightly better than the traditional RNN model in practice.

[i] www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks–1489512765771.html

[ii] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

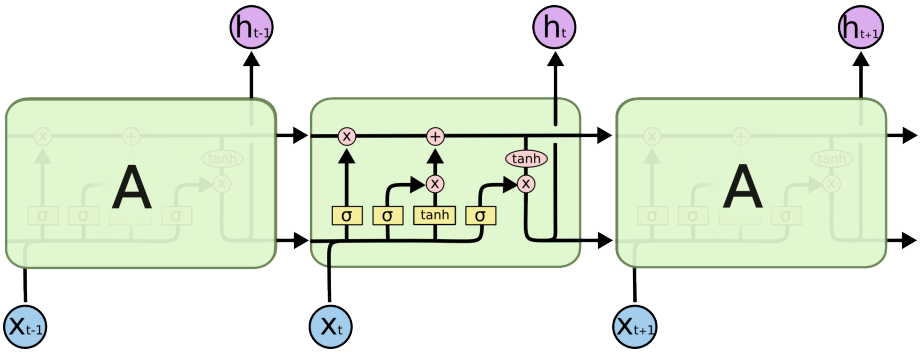


Figure 1.2: A simple structure of LSTM. (Source[ii])

Figure 1.2 shows a simple example of LSTM. All recurrent neural networks, including LSTM models, have the form of a chain of repeating modules of neural network (Figure 1.2 shows the module is repeated three times). An LSTM model takes corresponding input data at each time step. The key to LSTMs is the cell state (the horizontal line running through the top of Figure 1.2) that passes through the entire chain. Four gates (yellow boxes in Figure 1.2) provide the ability of an LSTM to add or remove information to the cell state according to the previous cell state and the current input. Thus, an LSTM model can extract useful features from every single step and maintain these features through the entire model. Using an LSTM model, Donahue et al. [37] proposed a scheme that yielded a good performance in the tasks of activity recognition, image description, and video description. Moreover, a Neural Image Caption model based on LSTM was proposed by Vinyals et al. [162] to automatically describe the content of an image. Zhang et al. [179] use an LSTM model to count vehicles in city cameras. All of these works prove that the LSTM Network has the capability to extract useful information from its inputs and generate distinguishing representations.

1.5 Contributions and Organization

In this dissertation, to fix the issues/limitations we discussed before, we make the following contributions in the field of human activity recognition, human motion intent prediction, and deep learning model acceleration.

- First, in Chapter 2, we propose a novel solution, namely SBGAR, for group activity recognition. It can be used to recognize single-person activity and group activity. The proposed

scheme is semantics-based. Specifically, we analyze the videos to generate descriptions describing the video frames, and then recognize the activities based on the semantic meaning of these descriptions. To the best of our knowledge, this is the first work that uses semantics to recognize human activities in videos. The experimental results show that our solution yields significantly better performance compared to the state-of-the-art approaches using two well-known datasets.

- Then, in Chapter 3, we enhance our SBGAR framework to a much faster model, namely ReHAR, by replacing the semantic representation with an intermediate probability distribution representation. The whole model is trained end-to-end to allow meaningful representations to be generated for the final activity recognition. In addition, ReHAR works better than SBGAR because ReHAR does not require a large dataset with semantic annotations of individual person's activity. Extensive evaluation using two popular activity datasets show that our scheme achieves higher accuracy and runs an order of magnitude faster than existing schemes. We also explore the visual explanation for our model to understand what it has learned.

- In Chapter 4, we propose a graph-based interaction-aware motion intent prediction scheme, called GRIP. GRIP uses a graph to represent the interactions of close objects, applies several graph convolutional blocks to extract features, and subsequently uses an encoder-decoder LSTM to make predictions. The experimental results on two well-known public datasets show that our proposed model improves the prediction accuracy of the state-of-the-art solution by 30%. The prediction error of GRIP is one meter shorter than existing schemes. Such an improvement can help autonomous driving cars avoid many traffic accidents. In addition, the proposed GRIP runs 5x faster than the state-of-the-art schemes.

- To further speed up the computation time, we propose a novel deep learning model decomposition solution in Chapter 5. The proposed solution replaces standard convolutional layers in a pre-trained model with separable layers to significantly reduce the number of floating-point operations (FLOPs). The decomposed model maintains high accuracy without using any data and training (fine-tuning) process. The experimental results on three computer vision

application scenarios show that DAC maintains high accuracies even when a vast amount of FLOPs is trimmed.

- Finally, we conclude this dissertation and discuss future works in Chapter 6.

Chapter 2

SBGAR: Semantics Based Group Activity Recognition

Designing a system that can recognize single-person activities and group activities is a challenging and important task. Most existing solutions [8, 44, 164, 14, 190] are proposed for single-person activity recognition. These solutions will perform worse in recognizing group activity since group activity recognition requires higher-level semantic features than single-person activity recognition. This weakness limits their widespread use, considering that group activities are more common than single-person events in real life.

Recently, some research has been done to recognize group activities from videos. Lan et al. [85] and Ramanathan et al. [127] used hand-crafted features in structured models to represent information between individuals in space and time domains. They, however, merely use shallow hand crated features and typically adopt a linear model that suffers from representation limitation. Lan et al. [87] believe that the contextual information of what other people in the scene are doing provides a useful clue for understanding high-level activities. Thus, they present a solution to recognize group activities by exploring group-person interaction and person-person interaction information. Based on the similar intuition that a strong correlation exists between a person's action and the actions of other nearby people, Choi et al. [23] exploit the spatial distribution of pedestrians in the scene as well as their pose and motion to achieve a robust action classification result. Next, they present a solution in [22] for simultaneously tracking multiple people and estimating

their collective activities. They introduce a hierarchy of activity types which correlates a specific person's action to the activity of the group. In addition, Ibrahim et al. [64] propose a hierarchical deep temporal model to infer group activities. Given a set of detected and tracked people, they run temporal deep networks (LSTMs) to analyze each individual person. They then aggregated these LSTMs over individual persons into a higher level deep temporal model. This allows the deep model to learn the relations between the people that contribute to recognizing a particular group activity.

Although these approaches achieve good performance in the task of group activities recognition, they are tracking-based methods. The biggest weakness of such approaches is their high computation time. These approaches have to first identify each individual person in video frames, track his/her individual activities, and then later infer the relationships of all people's activities before they can predict the group activity label and thus incur much computation.

To solve such weaknesses, instead of using time-consuming object detection and tracking models, we propose to represent global visual features extracted from input video frames as semantic descriptions, and then classify the group activity labels only based on these semantic descriptions. In Section 2.1, we describe our proposed group activity recognition scheme and implementation details. Our proposed solution [94] can be used for single-person activity recognition as well as group activity recognition. The experimental results reported in Section 2.2 show that our scheme yields much better performance and runs faster than the state-of-the-art approaches. Finally, we conclude this chapter in Section 2.3.

2.1 Proposed Scheme

Here, we present a novel model for recognizing group activities in videos. The intuition of our scheme is that people can easily infer an activity from a sequence of sentences. For example, given the following three sentences describing a volleyball game:

- **Sentence 1:** *"There is a player jumping on the right side, while others are standing."*
- **Sentence 2:** *"There is one player spiking on the right side and three players blocking on the left side, while others are standing."*

- **Sentence 3:** *"All players are standing."*

a person can easily infer that the right team is performing an offensive action (spiking) while the left team is playing a defensive action (blocking). Thus, we design a model which generates a caption for each frame in a video and then predicts the activity based on a sequence of generated captions. Figure 2.1 shows the architecture of our scheme which consists of three steps: input preprocessing, caption generation, and activity prediction.

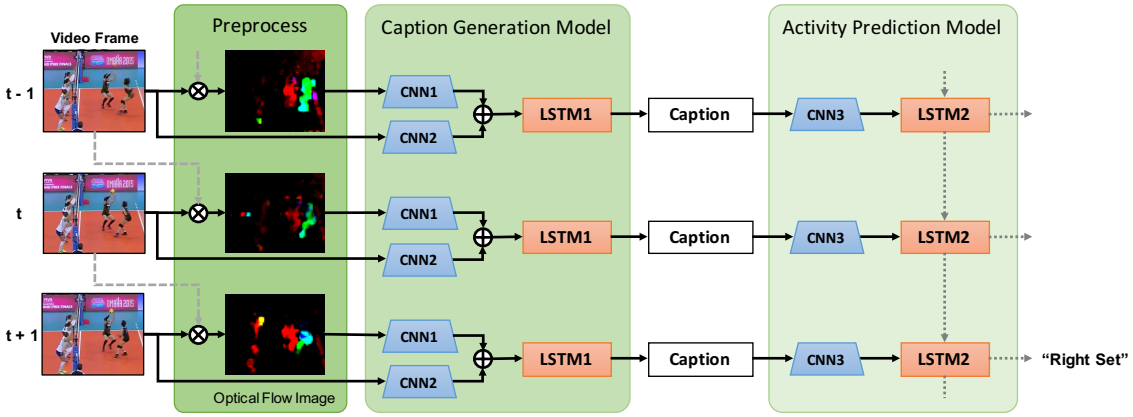


Figure 2.1: The architecture of the proposed Scheme. Caption Generation Model generates a caption to describe the corresponding frame. Activity Prediction Model is used to predict the group activity based on generated captions of a continuous sequence of frames. Symbol $\otimes$ indicates the operation of computing the dense optical flow image using two continuous frames, while symbol $\oplus$ indicates the operation of concatenating two CNN feature vectors into one single vector. In order to simplify the figure, the details of models are not shown here. Please refer to Figure 2.2 for more details of the Caption Generation Model, and Figure 2.3 for the Activity Prediction Model.

2.1.1 Preprocessing

We believe that both the scene features extracted from the original frames, and the movement features extracted using dense optical flow method [43] contribute towards group activity recognition. The original video frames contain more information about the environment, e.g., indoor or outdoor, while the derived optical flow images provide motion information. Thus, we use both types of features.

During preprocessing, we generate an optical flow image for every single video frame (except the first frame in a video). Given a video frame (frame t) as well as its previous one (frame $(t-1)$), we compute the displacement (direction and distance) of each pixel point in the frame. Then,

in HSV color space, we set the direction and distance corresponding to the Hue and value plane correspondingly, and set the saturation value to be a constant value, e.g. 255. The generated optical flow images are illustrated in Figure 2.1.

2.1.2 Caption Generation Model

After preprocessing, at time t, we have an original video frame and its corresponding dense optical flow image. We extract CNN features from both original frames (CNN2 in Figure 2.1) and optical flow images (CNN1 in Figure 2.1). Then, we concatenate CNN1 and CNN2 into a single vector.

Next, we build a Language Model using the LSTM model. There are two reasons why a LSTM model is used here: (1). A LSTM model can generate good captions using the CNN feature vector as its input [162]. (2). A LSTM model also helps us handle some scenarios in which we need to split the scene into different groups, e.g., a left and a right team in a volleyball game. Figure 2.2 shows the details of our model for caption generation.

During the training process: The inputs of the Caption Generation Model consist of (i) concatenated CNN Features, (ii) Input Captions, and (iii) Target Captions (Ground Truth). In this thesis, we encode each word of the Input Caption into a vector using One-Hot encoding. Considering One-Hot code is a high dimensional sparse feature which costs large storage and inefficient computation, we employ the word2vec model [115] to convert the One-Hot code into a continuous vector with a much lower dimension. We then feed the CNN Feature as well as the word2vec vector into an LSTM model (LSTM1) to generate the probability distribution of the next word in the sequence. Finally, the probability distribution will be compared to the Target Caption (Ground Truth) to tune the parameters of the model such that the predicted probability of the correct next word is higher than others.

Figure 2.2 shows the process when our model is fed the CNN Feature and the Input Caption "<SOS> A player is jumping" as the input, assuming the set of vocabulary is {"<SOS>", "A", "player", "is", "jumping", "<EOS>"}. The Output Layer contains the predicted probabilities that the LSMT1 assigns to the next word. The predicted result is "jumping A jumping is is", while the Target Caption is "A player is jumping <EOS>". It is obvious that such a prediction is not our expectation. Thus, we tune the parameters to increase the probability of the correct word (in red

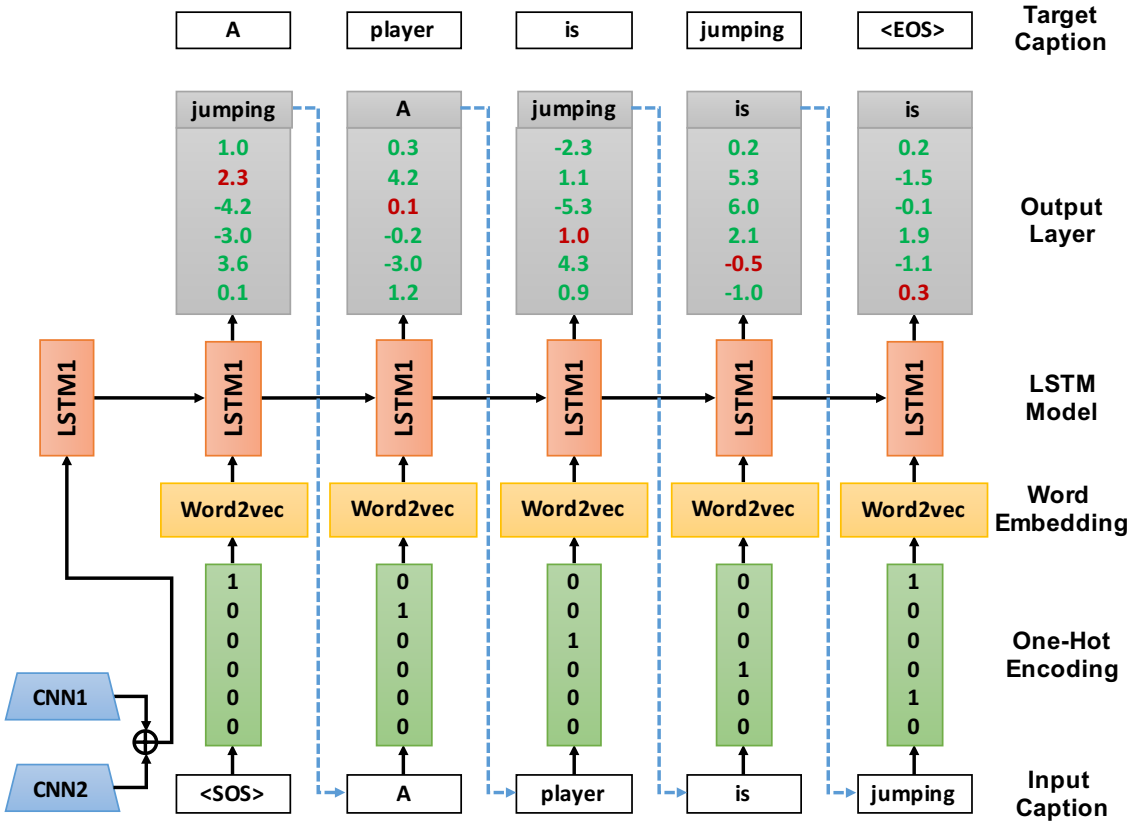


Figure 2.2: Caption generation model. <SOS> and <EOS> are symbols used to indicate the beginning and the ending of a caption correspondingly.

color) and decrease the probabilities of all other words (in green color). The process is repeated multiple times until the model converges and it can perform a good prediction.

During the testing process: The inputs of our model only consist of (i) CNN Features and (ii) Input Captions (initialized with a single starting symbol, <SOS>). The trained model, LSTM1, generates a probability distribution over what words are likely to come next. We then choose one word with the highest predicted probability and feed it right back into the model (blue dashed-line in Figure 2.2). This process is repeated many times until the predicted word with highest probability is the ending symbol, <EOS>, or the length of the generated caption is longer than a pre-determined threshold (e.g., 20).

2.1.3 Activity Prediction Model

The final step of SBGAR is to predict the activity label based on a sequence of generated captions using a LSTM model (LSTM2 in Figure 2.1). Instead of taking the captions as the input of the LSTM2 directly, we first employ a CNN model (CNN3 in Figure 2.1) to extract feature vectors from captions. The reason is threefold: 1. The lengths of generated captions vary while the input size of all cells in LSTM2 is the same. 2. A CNN model can generate vectors with the same dimension even if the lengths of input captions vary. 3. Kim et al. [77] show that a simple CNN model achieves excellent results in the task of sentence classification.

Figure 2.3 shows the details of our Activity Prediction Model. In this thesis, we use a similar network as in [77] which originally consists of 4 layers. We remove the last layer of the network in [77] and concatenate its first three layers with a LSTM Model (LSTM2) by taking the output of Layer 3 as the input of the LSTM2. Using a LSTM model to analyze a sequence of captions makes intuitive sense, considering how such a model resembles the way we process language: reading sequentially. The first three layers of the network in [77] are:

Layer 1: In this layer, we employ word2vec model [115] to convert an input caption into a matrix. Each row of the matrix corresponds to one word. In Figure 2.3, we show two input captions. One caption consisting of 8 words and another consisting of 4 words. The dimension of word2vec is set to 5, thus these two input captions are represented by two matrices (8*5 and 4*5 correspondingly).

Layer 2: The second layer performs convolutions over the word matrix using multiple filter sizes. In vision, the filters slide over local patches of an image, while in the field of National Language Processing (NLP), we typically slide the filters over the full rows of the word matrix considering each row represents a word. Thus, we set the dimension of the filters equals to the dimension of the word matrix. In Figure 2.3, we only show 2 filter sizes (2*5 and 3*5). The 2*5 filter will slide over 2 words each time, while the 3*5 filter will slide over 3 words each time. We perform convolution operation on both word matrices using two filters and end up with two feature maps for each word matrix.

Layer 3: In this layer, max-pooling is performed on each feature map. As shown in Figure 2.3, after max-pooling, both input captions (different lengths) are represented as two dimensional

features.

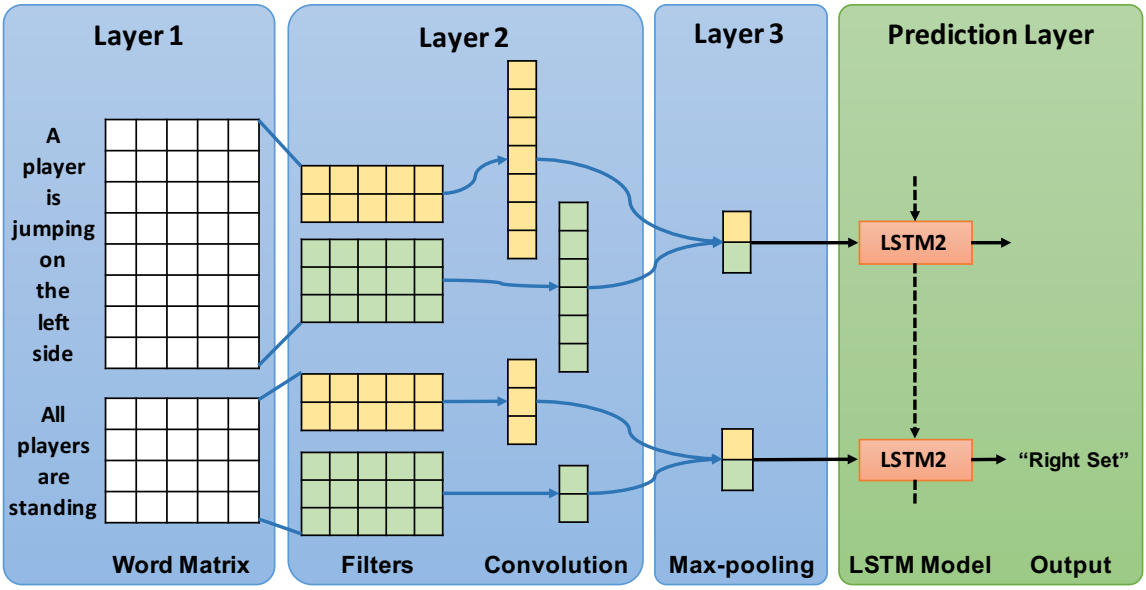


Figure 2.3: Activity prediction model.

In our SBGAR scheme, one caption is generated for each video frame and a sliding window of size $time\_steps$ frames is used to feed $time\_steps$ captions to LSTM2. For example, the window size shown in Figure 2.3 is 2 which means LSTM2 predicts an activity result based on 2 continuous input captions. By sliding the window, our model can analyze videos with varying number of frames.

During the training process: The inputs of our model consist of (i) a sequence of captions, more precisely $time\_steps$ captions, and (ii) their corresponding activity labels (Ground Truth). Given $time\_steps$ captions, our model first extracts CNN features from these captions and then feed the CNN features into the prediction layer (LSTM2) to generate a probability distribution for all potential labels. We then compute the mean value of the cross-entropy loss, as shown in Equation (2.1). The purpose of the training process is to minimize such a loss function L, where N is the size of the training set, y are the ground truth labels, and p are the predicted probabilities. During each training iteration, we tune the parameters of our model based on the value of this loss function L. We repeat feeding training captions and their corresponding labels to train our model until the value of L becomes smaller than a pre-determined threshold or the number of iterations is larger than another pre-determined threshold.

$$L(y,p) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log p_i \tag{2.1}$$

During the testing process: The inputs of our model will only consist of $time\_steps$ captions. After the model generated a probability for all activity labels, we choose the one with the highest probability as the final result.

2.1.4 Implementation Details

We implement our scheme using Python Programming Language and Tensorflow [1]. Tensorflow is an open source software library for machine learning released by Google. We report the implementation details of our scheme and the settings of important parameters as follows.

CNN1 and CNN2: To extract CNN features from images, we use an Inception-v3 model [153] pre-trained on ImageNet [31] as a feature extractor. Specifically, we use the output of the final pooling layer (pool_3) in Inception-v3 model as the CNN feature of an image. Thus, the dimension of the extracted CNN feature is 2048 and the dimension of concatenated CNN1 and CNN2 features is 4096.

LSTM1: The LSTM1 is a 2-layer LSTM Model having 1024 hidden units. Before feeding the captions into the LSTM1, we use word2vec model [115] to convert each caption into a dense representation with a low dimension. We set the embedding size to 1024, thus the size of embedded captions is $n_w * 1024$, where n_w is a length of a caption (sort caption are zero-padded). Because the size of all cells in a LSTM model are the same, so we use a transformation matrix $(4096 * 1024)$ and a bias vector $(1 * 1024)$ to transform a $1 * 4096$ CNN feature into $1 * 1024$ (1024 is the dimension of embedded captions). To do so, we only need to multiply the CNN feature $(1 * 4096)$ with the transformation matrix $(4096 * 1024)$ and add the bias vector $(1 * 1024)$. Then, we concatenate the transformed CNN feature with the embedded caption and feed them$((n_w + 1) * 1024)$ into LSTM1. **During the training process:** We set the learning rate to $1e^{-4}$ initially and reduce the learning rate every epoch until it reaches $1e^{-6}$. In order to reduce overfitting, we use the dropout technique [148] and set the input & output keep probabilities to 0.75. **During the testing process:** The input caption is initialized with a starting symbol (<SOS>). We set the maximum length of the generated caption to be 20. The input and output keep probabilities are set to 1 to disable dropout.

CNN3: We first embed the generated captions into a dense representation using word2vec model before feeding them into the CNN model (CNN3). We set the embedding size to 5, so the size of the embedded caption is $n_w * 5$, where n_w is the length of a caption. Instead of using a pre-trained CNN model, we implement a simple CNN model which only performs convolution and max-pooling operations with a generated caption as its input. Four filter sizes [3*5, 4*5, 5*5, 6*5] are used with 5 filters for each size. Thus, there is a total of 20 filters in this CNN model. Each filter slides over the whole embedded caption using a VALID Padding Method (VALID padding means there will be no zero padding outside the edges when we do max pool). Once we have all the max-pooled outputs from each filter, we combine them into one long feature. Thus, the length the feature generated by CNN3 is 20.

LSTM2: The LSTM2 is a 2-layer LSTM model. The sequence length of LSTM2 is set to 10, which means the LSTM2 will analyze 10 captions each time. **During the training process:** The learning rate is set to $1e^{-4}$ initially and reduced each epoch until $1e^{-6}$. We use the Adam algorithm [78] to minimize the cost function. To avoid overfitting, we employ the dropout method [148] and set the input and out keep probabilities to 0.75. **During the testing process:** The input and output keep probabilities are set to 1.

2.2 Experiments

We run our scheme on a desktop running Ubuntu 14.04 with 4.0GHz Intel Core i7 CPU, 16GB Memory, and NVIDIA Geforce GTX 1080 Graphics Card.

2.2.1 Datasets

We evaluate our scheme using two datasets: Collective Activity Dataset [23] and Volleyball Dataset [64].

Collective Activity Dataset: The Collective Activity Dataset has been widely used to evaluate the performance of group activity recognition schemes. It consists of 44 videos clips acquired using a low resolution hand-held camera. The location, action, and pose of each person in the videos is labeled. The five action categories include: crossing, waiting, queuing, walking, and talking while the pose categories include: right, front-right, front, front-left, left, back-left, back, and back-right.

Thus, we trained the classifier to predict these five group activity categories depending on what the majority of the people included in the videos are doing: crossing, waiting, queuing, walking, and talking. Pose information is not used in our scheme.

Volleyball Dataset: The Volleyball Dataset was released by Ibrahim et al. [64] to evaluate the performance of group activity recognition schemes on sport footage. All videos related to volleyball games are collected from YouTube. In total, there are 1525 frames labeled with seven player action labels (waiting, setting, digging, falling, spiking, blocking, and others) and six group activity labels (right set, right spike, right pass, left pass, left spike, and left set). The location of each player is also labeled and that information is not used in our scheme.

2.2.2 Metrics

In order to compare our scheme with Ibrahim et al. [64], we use the same metrics used in [64].

Classification Accuracy: The accuracy is the percentage of the correct predictions.

Confusion Matrix: A confusion matrix [79] contains information about actual and predicted classifications generated by a classification system. In a confusion matrix, each column represents the instances of an actual class, while each row represents the predicted classes.

2.2.3 Baselines & SGBAR

In this section, we want to compare the following baselines and SGBAR with some existing schemes proposed by other researchers.

B1. Single Frame Classification: B1 fine-tunes the Inception-v3 model for group activity recognition based on a single frame.

B2. Temporal Model with Image Features: B2 is the solution proposed by Donahue et al. [37] where the image feature is extracted from the final pooling layer (pool 3:0) of Inception-v3 model and fed directly to a 2-layer LSTM model to recognize group activities.

B3. SBGAR (RGB Frame Only): B3 is a variant of our SBGAR scheme which only considers the RGB frames as the input ignoring any extracted optical flow information.

B4. SBGAR (Optical Flow Image Only): B4 is another variant of our SBGAR scheme which only considers the optical flow information while ignoring the information extracted from the RGB

frames.

SBGAR (RGB Frame & Optical Flow Image): SBBAR considers information from both the RGB frame and optical flow image.

Comparing B1 & B2 allows us to see how much improvement can be obtained using a group of frames for group activity recognition. Similarly, comparing B3, B4 & SBGAR allows us to evaluate the improvement that can be achieved by combining both the scene and the motion related information.

2.2.4 Experiments on the Collective Activity Dataset

In this subsection, we report our experimental results using the Collective Activity Dataset. In order to train the caption generation model (LSTM1), we manually labeled a caption for each training frame. Instead of generating complete sentences, we generate captions only using important keywords. The reasons are threefold. (1). Training a model that can generate whole sentences from images requires a large amount of labeled data. However, there is no public dataset that provides such annotated sentences for the human activity recognition task. (2). Our purpose is to recognize group activities based on captions rather than generating complete sentences. Thus, our scheme will work as long as LSTM1 can generate several useful words. (3). Training a language model which can generate complete sentences incurs longer time, because it needs to learn the grammar which is not useful for activity recognition. Considering that this dataset contains the location and individual action of every person in each video frame, we can easily label captions for the actions of all players in the training frames as follows:

"<SOS> Walking Crossing Crossing Crossing <EOS>"

"<SOS> Waiting Waiting Waiting Crossing Walking <EOS>"

In Table 2.1, we report our experimental results (accuracy) using the Collective Activity Dataset and compare our SBGAR related and baseline methods with other existing methods. In [64], the authors compare their scheme with Contextual Model [87], Deep Structured Model [33], and Cardinality kernel [55] using the Collective Activity Dataset. Thus, we include the results they reported in Table 2.1. We follow the same experimental settings as used in [64], i.e., 1/3rd of the video clips were selected for testing and the rest for training (video clips for training and testing are

selected from different videos). During the SBGAR related training process, we use 500 epochs to train the LSTM1 model and 300 epochs to train the LSTM2 model. For the LSTM2 model, we predict the final activity result based on a window size of 10 frames (5 before, current and 4 after frames, corresponding to 0.4 second activity) (the same setting as [64]).

Methods	Accuracy (%)
B1 - Single Frame Classification	67.2
B2 - Temporal Model with Image Features	68.5
B3 - SBGAR (RGB Frame Only)	83.7
B4 - SBGAR (Optical Flow Image Only)	70.1
Contextual Model [87] *	79.1
Deep Structured Model [33] *	80.6
Two-stage Hierarchical Model [64] *	81.5
Cardinality kernel [55] *	83.4
SBGAR (RGB & Optical Flow)	**86.1**

Table 2.1: Comparison of our scheme with baseline methods and previously published works on the Collective Activity Dataset. The results for "*" were extracted from [64].

The experimental results in Table 2.1 show that our proposed scheme outperforms the baseline methods as well as other existing schemes. It is worth pointing out that even when we only use a single feature (baseline B3), our proposed scheme can still achieve a higher accuracy than the state-of-the-art method in [55].

The baseline method B3 achieves a higher accuracy than B4 because most people in the videos in this dataset hardly move while they are talking, waiting, or queuing, which means not much useful information can be extracted from the optical flow analysis of these videos for activity recognition. B3 uses the information extracted from RGB frames and hence performs better.

Figure 2.4 shows the comparison of the confusion matrices between the scheme in [64] and SBGAR using the Collective Activity Dataset. From this figure, one can see that [64] predicts some instances belonging to "crossing" and "waiting" as "walking", while SBGAR reduces this error. However, both [64] and SBGAR can not easily distinguish between "crossing" and "waiting". There are two reasons: 1. "crossing" and "waiting" often happen in the same scene, e.g. "at a cross road". 2. These two activities often happen sequentially, e.g. one waits at a cross road first, and then crosses. We notice that, comparing to [64], SBGAR predicts some "talking" instances as "walking". We discover that some video clips contain both activities and SBGAR believes that

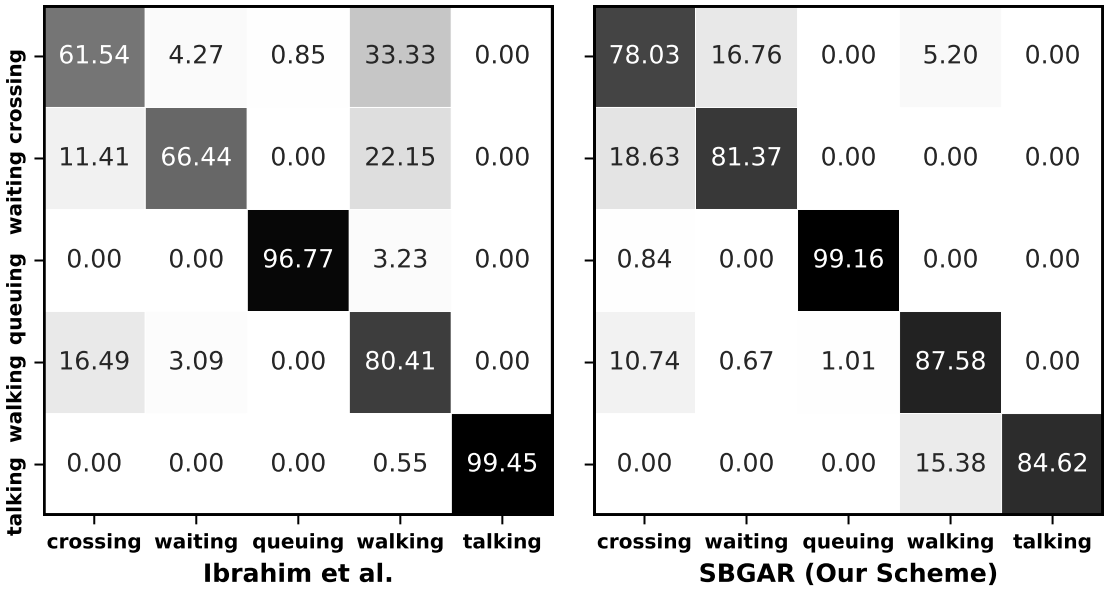


Figure 2.4: Comparison between [64] (left) and SBGAR (right) on the Collective Activity Dataset.

"walking" activity is more obvious than "talking" in these video clips.

2.2.5 Experiments on the Volleyball Dataset

In this subsection, we report our experimental results on the Volleyball Dataset. Based on the target activity labels (Left pass, Left set, Left spike, Right pass, Right set, Right spike), we notice that the labels contain information regarding whether the players are in the left or right side of the court, which means that we need to divide players into two groups. To handle this application scenario, we adjust the captions. We manually labeled captions for all training frames as follows:

"<SOS> Left: waiting moving blocking Right: standing spiking <EOS>"

"<SOS> Left: standing blocking Right: standing spiking <EOS>"

The order of the words describing the actions of each individual team is arbitrary. To make the training phase more efficient, we keep the order of the actions taken by both sides static (i.e. actions from the left are listed first). In Table 2.2, we report our experimental results (accuracy) using the Volleyball Dataset and compare the baseline and SBGAR related methods with existing methods. Two third's of the video frames are used for training, and the remaining 1/3rd for testing (the same setting as Ibrahim et al. [64]). For SBGAR related methos, we use 500 epochs to train the LSTM1 model and 300 epochs to train the LSTM2 model. For the LSTM2 model, we predict

the final activity result based on an observation window of 10 frames (5 before, current, and 4 after frames, corresponding to 0.4 second activity) (the same setting as in [64]).

Methods	Accuracy (%)
B1 - Single Frame Classification	41.9
B2 - Temporal Model with Image Features	44.3
B3 - SBGAR (RGB Frame Only)	38.7
B4 - SBGAR (Optical Flow Image Only)	54.3
Two-stage Hierarchical Model [64]	51.1
SBGAR (RGB & Optical Flow)	**66.9**

Table 2.2: Comparison of our scheme with baseline methods and previously published works on the Volleyball Dataset.

The experimental results show that our proposed SBGAR scheme outperforms the baseline methods and the state-of-the-art methods [64] on this dataset. It is worth pointing out that B4 (only a single feature is used) achieves a better result than [64].

For this dataset, B4 performs better than B3 by 15.6% in terms of achieved accuracy because the videos in the Volleyball dataset have the same scene (same viewpoint, similar background, similar color, etc) and hence fewer distinguishing features can be extracted in B3. However, B4 can extract more meaningful features (motion information) from the optical flow images.

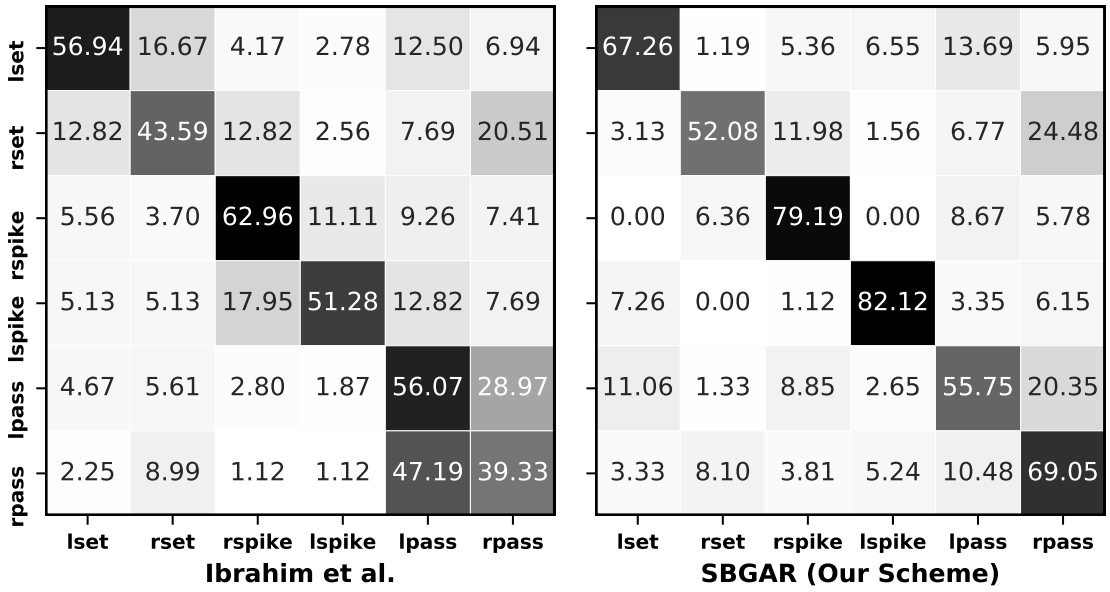


Figure 2.5: Comparison between [64] (left) and SBGAR (right) on the Volleyball Dataset.

Figure 2.5 shows the comparison of the confusion matrices between [64] and SBGAR on the Volleyball Dataset. It is clear that SBGAR achieves a better result in distinguishing activities which take place at the left and right side of the court, especially "lspike" versus "rspike". This improvement comes from the fact that our proposed model can generate captions for both the left and right parts. To a certain extent, the experimental results prove that our Caption Generation Model has the ability to consider the spatial information and represents such information in the generated captions correctly. However, we notice that both [64] and SBGAR predict some "lset" and "rset" samples as "lpass" and "rpass" correspondingly. This is because those "set" and "pass" activities are similar and often appear in the same region within a court from the view of the camera.

2.2.6 Impact of Key Parameters

The settings of parameter values have an impact on the predicted results of a Machine Learning model. Thus, we evaluated the impact of two key parameters:

Epochs: Each epoch is defined as the process of feeding the whole training set to a model. In SBGAR, we use two models, Caption Generation Model (LSTM1) and Activity Recognition Model (LSTM2). Thus, we will evaluate the impact of the number of epochs on their accuracy during the training of both models.

Observation Window Size of LSTM2: The observation window size is defined as the number of video frames that are used to generate a prediction. If the window size is 5, it means that LSTM2 will generate a prediction based on 5 consecutive frames.

We discuss the details as follows:

1. Epochs for LSTM1: In Figure 2.6, we report the accuracy of SBGAR on both datasets as we fix the number of training epochs of LSTM2 to 300 while varying the number of training epochs of LSTM1. The solid curve in blue color is the result using the Collective Activity Dataset, while the dashed curve in green color is the result using the Volleyball dataset. One can observe that larger epochs lead to higher accuracy. The accuracy becomes stable when the number of epochs exceeds 500 for both datasets. Figure 2.7 shows the training loss as we varies the number of epochs during the training process of LSTM1. The blue line with "*" marker shows the training loss, while the solid red line shows the testing loss. The training and testing losses decrease as the number of

epochs increases and approach a stable value after 400 epochs.

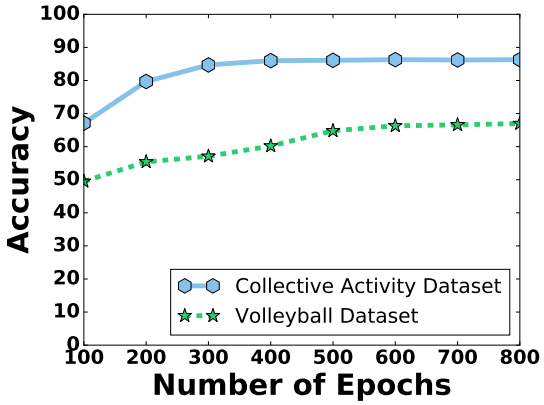


Figure 2.6: Activity recognition accuracy as the number of training epochs of LSTM1 is varied.

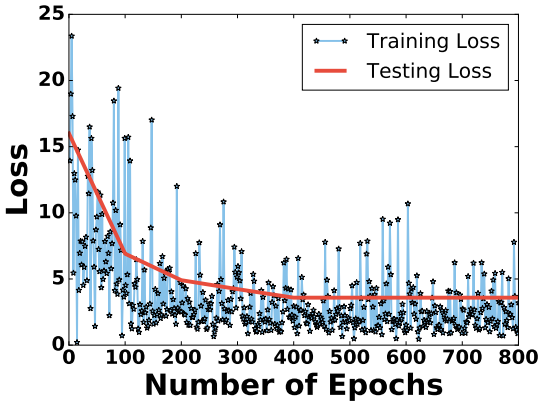


Figure 2.7: LSTM1 training loss as the number of training epochs is varied using the Collective Activity Dataset.

Based on both observations, we choose 500 as the number of epochs for training LSTM1.

2. Training epochs of LSTM2: In Figure 2.8, we report the testing accuracy of SBGAR on both datasets as we fix the number of training epochs for LSTM1 to 500 while varying the number of training epochs for LSTM2. The solid curve in blue color is the result using the Collective Activity Dataset, while the dashed curve in green color is the result using the Volleyball dataset. One can see the accuracy increases as the number of epochs increases and becomes stable after 200 epochs. Figure 2.9 shows the training loss on the Collective Activity dataset as we increase the number of epochs during the training process of LSTM2. The training and testing losses decrease as the number of epochs increases and become stable after 300 epochs.

Based on both above observations, we choose 300 as the default number of epochs for training LSTM2.

3. Observation Window Size of LSTM2: For video based activity recognition, only using frames before the current frame seems to make more sense in real life, considering that one can not access the frames after the current frame. A model which predicts a correct result only based on the previous frames may have the capability of early detection. Such a model is more useful for early-warning systems. However, adding some frames after the current frame may improve the prediction performance because more frames means more useful information can be used in the prediction process. Taking the volleyball sport as an example, assuming that a player is jumping,

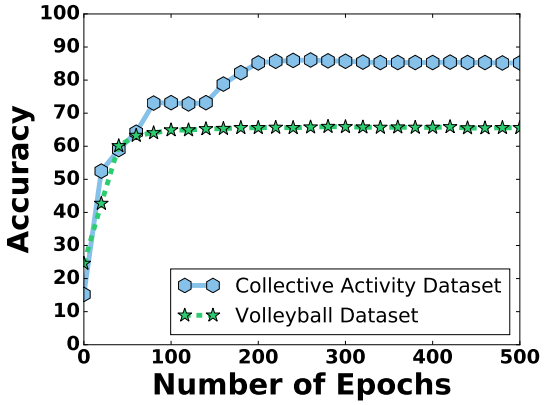


Figure 2.8: Activity recognition accuracy as the number of training epochs of LSTM2 is varied using both datasets.

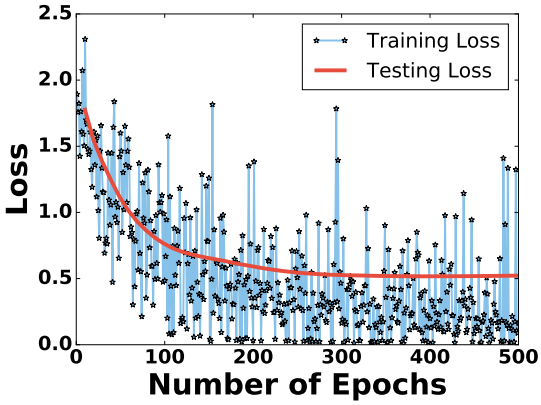


Figure 2.9: LSTM2 training loss as the number of training epochs is varied using the Collective Activity datasets.

it is hard to say whether the player is "blocking" or “spiking” only based on this observation. By observing more frames, one can predict a more accurate result. Even though a model using “future" frames incurs additional delay, such a model may be more useful in some application scenarios.

In order to evaluate the performance of SBGAR with varying length of input frame sequence, we consider the following types of input frame sequences:

Before(x): x frames before the current frame are used as the input sequence.

After(x): x frames after the current frame are used as the input sequence.

Before(x)After(y): x frames before and y frames after the current frame are used as the input sequence.

Frame Sequences	**Accuracy (%)**	
	Collective Activity Dataset	**Volleyball Dataset**
Before(10)	85.7	64.7
Before(5)	84.1	64.7
After(5)	83.6	65.1
After(10)	84.7	65.1
Before(5)After(5)	86.1	66.9
Before(5)After(10)	85.9	67.4
Before(10)After(5)	86.3	67.1
Before(10)After(10)	**86.4**	**67.7**

Table 2.3: Accuracy on both datasets by taking variant input frames.

We report the experimental results in Table 2.3. One can easily notice that using a larger win-

dow size helps to improve the accuracy. In addition, comparing to only using frames before or after the current one (top 4 rows), using frames before and after the current frame (tail 4 rows) achieves a higher accuracy on both datasets. If we focus on the results of "Before(10)" and "After(10)", we can discover that "Before(10)" produces a better result on the Collective Activity Dataset, while "After(10)" performs better on the Volleyball Dataset. The same observation can be made between "Before(5)After(10)" and 'Before(10)After(5)". The reason of this is threefold: 1. The activities in the Collective Activity Dataset are more constant, which means there is no big differences between two continuous frames if they share the same activity, e.g. walking or queuing. 2. A video clip in the Collect Activity Dataset contains several activities, e.g. crossing and walking may happen alternately. 3. Activities in the Volleyball Dataset may involve the same action in their beginning frames, e.g. both blocking and spiking involve jumping. Thus, adding some frames after the current frame may cause a wrong prediction result in the Collective Activity Dataset, while it helps in the Volleyball Dataset.

2.2.7 Computation Time

For some application scenarios, e.g., sport analytics, it is highly important to be able to predict a group activity label in real time. Thus, we are interested in comparing the computation time of our scheme and the state-of-the-art group activity recognition scheme [64]. In Table 2.4, we report details of the computation time of our scheme. All data are averaged over 5 runs on the Volleyball dataset. With a sliding window of 10 frames, our scheme can predict on the average a group activity label within 108.5ms. If we use non-overlapping window of 10 frames, our scheme only takes about (22.19+27.78*2+28.63)*10+2.15=1065.95 ms (1.066sec). Running the code released by the authors in [64] using the same machine, the prediction time takes 4.22 seconds without including the time it takes to detect individual players. Thus, our scheme will be more useful for real-time prediction of group activity.

2.3 Discussion

In this chapter, we propose a novel scheme (SBGAR) to recognize group activities in videos. The proposed method generates a caption for each video frame first, and then predicts the final activity

Process (Based on Single Frame)	Computation Time (ms)
Optical Flow Image	22.19
Extract CNN1 Feature (Inception-v3)	27.78
Extract CNN2 Feature (Inception-v3)	27.78
Caption Generation	28.63
Activity Recognition (Based on 10 Frames)	2.15
In Total	108.53

Table 2.4: Computation time of SBGAR.

categories based on these generated captions. The experimental results on two well-known datasets demonstrate the effectiveness and accuracy of our proposed method. Compared to the existing state-of-the-art methods [87, 23, 22, 64], our scheme achieves a higher recognition accuracy with a shorter computation time. The reason that our SBGAR outperforms the existing methods is two-fold: (1) Our SBGAR does not use time-consuming models, e.g., object detection or tracking models, that benefits our scheme runs much faster than the state-of-the-art solutions. (2) SBGAR innovatively uses semantic descriptions to generate a high-level representation for each input video frame. This helps the model (especially for the second LSTM) to predict group activities at a high semantic level, which is important for the group activity recognition task. Although we did not train the model to generate whole sentences because of the lack of annotated complete sentences in the datasets we used, our proposed scheme has similar structures as in [37, 161] which are proposed for image caption generation and hence we believe our scheme will have similar capability of generating whole sentences if large datasets with annotated complete sentences are available. Our SBGAR generates an independent semantic representation for each input video frame, that helps improve the robustness of the model. Specifically, when the model does not precisely generate a semantic representation at a time step, our SBGAR can still correctly predict activities based on other semantic representation. However, the existing models do not work well once the object detector or tracker generates a wrong result, e.g., wrong detection or lost tracking.

Chapter 3

ReHAR: Robust and Efficient Human Activity Recognition

In the previous chapter, we proposed a semantics-based group activity recognition scheme that uses an LSTM model to generate a caption for each video frame and another LSTM model to predict the final activity categories based on the generated captions. Although it achieves a higher accuracy with a shorter running time, it has three weaknesses: (1) the caption generation model cannot always generate a perfect caption; (2) the caption generation model is trained only based on its own loss without getting any feedback from the final output of the model. This will result in lower accuracies since the generated captions may not contain useful information for the second model to predict the final activities; (3) it is very difficult to access a large dataset which contains caption information, e.g., individual action annotations.

To remove such limitations, we explore an end-to-end trainable model [95] that can overcome these weaknesses in this chapter. In Section 3.1, we describe our proposed activity recognition scheme and implementation details. We report our experimental results in Section 3.2. Finally, we conclude this chapter in Section 3.3.

3.1 Proposed Scheme

We propose an end-to-end model for recognizing activities in videos. The intuition of our model is that if we can generate a good representation for every single frame, then it is easier for the model to infer the final activity label for the whole video based on these representations. The model, illustrated in Figure 3.1, consists of three components: (1) Input Preprocessing Model, (2) Single Frame Representation Model, and (3) Activity Recognition Model.

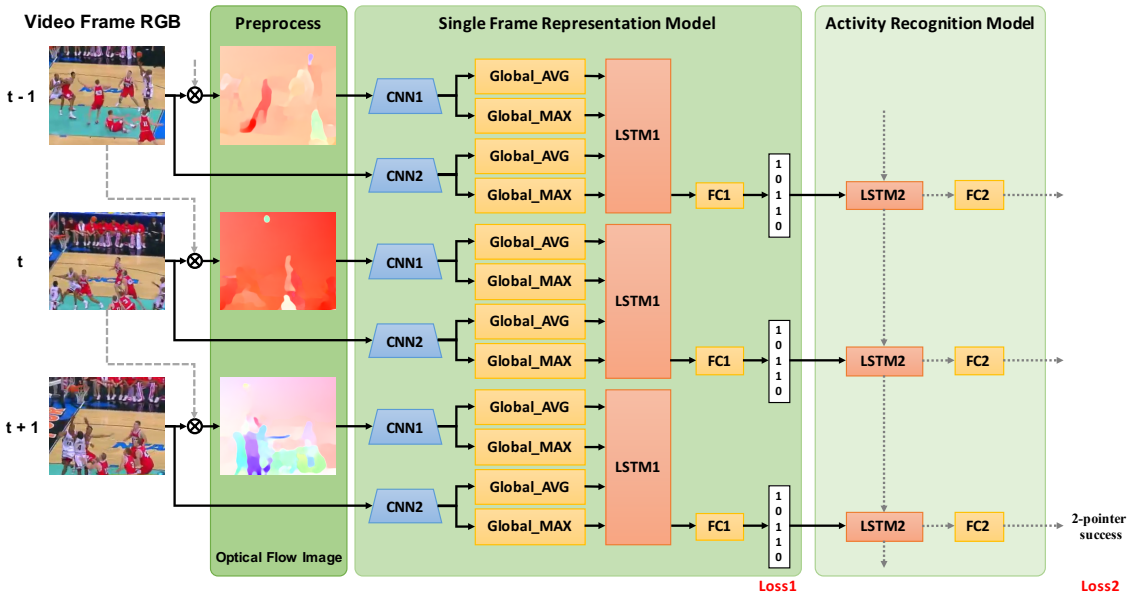


Figure 3.1: The architecture of the proposed Scheme. Symbol $\otimes$ indicates the operation of computing the dense optical flow using two continuous frames. CNN1 and CNN2 indicate VGG16 (layer "block1_conv1" to layer "block5_pool").

3.1.1 Preprocessing

We argue that both the background context and the motion of people contribute towards the group activity recognition. Thus, we also use the original frames (contain environment information) and their corresponding optical flow images (provide motion information) in our scheme. During the preprocessing phase, we feed a video frame (at time t) and its previous one (at time $t-1$) to the FlowNet 2.0 [66] to compute optical flow, since FlowNet 2.0 provides the best performance for generating optical flow. Then, we use the method described in [7] [i] to visualize the optical flow

[i] http://vision.middlebury.edu/flow/

information into a colorful image (3 channels), namely optical flow image. We generate an optical flow image for every frame (except the first one) in a video. The generated optical flow images are illustrated in Figure 3.1.

3.1.2 Single Frame Representation Model

The Single Frame Representation Model consists of two CNN feature extractors (one for video frame and another for optical flow image) and an LSTM model. Although any CNN model can be used as a feature extractor in our model, to simplify the explanation, VGG16[ii] [145] is used in this section and the size of the video frames and the optical flow images are fixed to (224x224x3).

Once we get the optical flow image at time t, we feed it as well as the corresponding video frame to two CNN models ("CNN1" and "CNN2" correspondingly in Figure 3.1) to extract features. Considering that, compared to FC layers, Global Pooling (GP) layers have at least two advantages: (1) The GP layers enforce correspondences between feature maps and categories, thus the feature maps can be easily interpreted as categories confidence maps. (2) There is no parameter to optimize in a GP layer, thus overfitting is avoided at this layer. Thus, instead of only removing the last prediction layer from VGG16 as in [37], we remove the last 4 layers (flatten, fc1, fc2, and prediction) and add a Global Average Pooling layer [100] and a Global Maximum Pooling layer [100] ("Global_AVG" (1x512) and "Global_MAX" (1x512) respectively in Figure 3.1) to its end. Thus, the "CNN1" and "CNN2" model in Figure 3.1 include layers from "block1_conv1" to "block5_pool" of VGG16. The layer "block5_pool" has (7x7x512) output size.

After that, we feed the output of these global pooling layers to an LSTM model (LSTM1 in Figure 3.1), which means the LSTM model has 4 input steps and each step has 512 dimensions. This LSTM model is used to extract useful features from 4 different input steps (2 outputs of Global Average Pooling layers and 2 outputs of Global Maximum Pooling layers). A fully-connected layer (FC1 in Figure 3.1) with a "softmax" activation function is added to the output of the final step of the LSTM1 to generate the representation for each input video frame.

[ii] VGG16: [block1_conv1, block1_conv2, block1_pool, block2_conv1, block2_conv2, block2_pool, block3_conv1, block3_conv2, block3_conv3, block3_pool, block4_conv1, block4_conv2, block4_conv3, block4_pool, block5_conv1, block5_conv2, block5_conv3, block5_pool, flatten, fc1, fc2, prediction]

3.1.3 Activity Recognition Model

The model predicts the final activity label based on a sequence of generated single frame representations. The Activity Recognition Model is an LSTM network (LSTM2 in Figure 3.1) that takes the single frame representations as its input. Thus, the input step $time\_step$ of LSTM2 equals to the number of the current input video frames. In Figure 3.1, $time\_step = 3$. Then, the output of the final step of the LSTM2 is fed into a fully-connected layer (FC2 in Figure 3.1) with a "softmax" activation function to predict the final activity label.

3.1.4 Implementation Details

Our scheme is implemented using Python Programming Language and Keras Library [24] with Tensorflow [1] backend. We report the implementation details of our scheme and the settings of important parameters as follows.

Optimization: We train our model as a multi-task learning. The overall loss can be computed as:

$$Loss = (\sum_{t=1}^{time\_step} loss_{1,t}) + \lambda * loss_2 \tag{3.1}$$

where $time\_step$ is the number of frames based on which the model predicts the final activity label (in Figure 3.1, $time\_step = 3$), $loss_{1,t}$ is the loss of the generated single frame representation at time t and $loss_2$ is the loss of prediction of the final activity. λ is the parameter that is used to balance the single frame representation generation loss and the final activity classification loss. In our experiment, we set $\lambda = 2$ to assign a higher weight to the final activity prediction, considering that the final activity prediction is our final purpose. The model is trained to minimize the $Loss$.

Single Frame Representation Model: The LSTM1 is a single layer LSTM with 200 hidden units. For FC1 layer, we set the dimension of its output to the number of the final activities and its training ground truth to be the one-hot vector of the final activity label. We train the Single Frame Representation Model as a classification task. In this case the representation is the probability distribution of each video frame over all activities. Thus, the $loss_1$ at time t, donated as $loss_{1,t}$,

can be computed using categorical cross entropy loss:

$$loss_{1,t} = -\sum_{i=1} g_{t,i} \log\left(p_{t,i}\right) \tag{3.2}$$

where g are the ground truth and p are the predictions. During the testing phase, the model will generate a probability vector as a representation for each frame.

Activity Recognition Model: The LSTM2 is also a single layer LSTM with 200 hidden units. The output of the FC2 is set to the number of categories. To train the model for a classification task, we train the $loss_2$ using categorical cross entropy loss:

$$loss_2 = -\sum_{i=1} g_i \log\left(p_{ti}\right) \tag{3.3}$$

where g are the ground truth and p are the predictions.

Training Process: To speed up the training process and get a better performance, we load the pre-trained VGG16 weights on Imagenet dataset [31]. We train the model using "rmsprop" optimizer with 0.001 learning rate and 1e-8 fuzz factor until the loss becomes converged. Then, we switch the optimizer to SGD with 0.0001 learning rate. The "rmsprop" optimizer helps the model converge quickly, and the SGD with a small learning rate helps to tune the model.

3.2 Experiments

We run our scheme on a desktop running Ubuntu 14.04 with 4.0GHz Intel Core i7 CPU, 128GB Memory, and a NVIDIA GTX 1080 Graphics Card.

3.2.1 Datasets

We evaluate our scheme on two well known activity recognition datasets: NCAA Basketball Dataset [128] and UCF Sports Action Dataset [132].

NCAA Basketball Dataset: The NCAA Basketball Dataset[iii] was collected by Ramanathan et al. [128] to evaluate the performance of activity recognition schemes on multi-person action videos.

[iii] http://basketballattention.appspot.com/

It is a subset (257 Basketball Game videos) of the 296 NCAA games available from YouTube[iv]. All videos are randomly split into 212 training, 12 validation and 33 testing videos. Each of these videos are split into 4 second clips and sub-sampled to 6fps. They filter out clips which are not profile shots, which results in a total of 11436 training, 856 validation, and 2256 testing video clips. Each of these video clips is manually labeled as one of these 11 labels: 3-pointer success, 3-pointer failure, free-throw success, free-throw failure, layup success, layup failure, other 2-pointer success, other 2-pointer failure, slam dunk success, slam dunk failure or steal success. The Basketball Dataset also annotates the bounding boxes of all the players in a subset of 9000 frames from the training videos. In our scheme, we do not use this location annotation.

UCF Sports Action Dataset: The UCF Sports dataset[v] [132] consists of a set of actions collected from a wide range of stock footage websites including BBC Motion gallery and GettyImages. It consists of a total of 150 videos. Each video has one of these 10 action categories: diving, golf swing, kicking, lifting, riding horse, running, skateboarding, swinging-bench, swinging-side, and walking.

3.2.2 Metrics

Mean Average Precision (mAP): Mean Average Precision is the mean of the average precision (AP) scores for each classification category. By computing a precision and recall, one can plot a precision-recall curve, plotting precision $p(r)$ as a function of recall r. Average precision computes the average value of $p(r)$ over the interval from $r = 0$ to $r = 1$ (please refer to wikipedia.org[vi]):

$$AP = \int_0^1 p(r)dr \tag{3.4}$$

Confusion Matrix: A confusion matrix [79] contains information about actual and predicted classifications generated by a classification system. In a confusion matrix, each row represents the predicted classes, while each column represents the instances of an actual class.

[iv] https://www.youtube.com/user/ncaaondemand
[v] http://crcv.ucf.edu/data/UCF_Sports_Action.php
[vi] https://en.wikipedia.org/w/index.php?title=Information_retrieval

3.2.3 Experiments on the NCAA Basketball Dataset

In this section, we report our experimental results on the NCAA Basketball Dataset. As described in [128], we classify isolated video clips into 11 classes without using any additional negative from other parts of the basketball videos. In this dataset, each video clip has 24 frames (6fps for 4 seconds). We use all frames for training and testing our model as done in [128]. The results are reported in Table 3.1. Among all 11 categories, our scheme achieves the highest accuracy at 8 categories compared to other baseline models. Overall, our scheme shows a 7.3% accuracy improvement compared to [128] (Atten. track in Table 3.1). We notice that all methods perform much poorer for categories such as "slam dunk failure". This is because we have very little data (47 training samples and 5 testing samples) belonging to "slam dunk failure" category in the Basketball dataset. The performance is much better for "free-throw" and "3-pointers", because these events have fixed and more obvious patterns (especially for "free-throw") and more training data in this dataset.

	3point S.	3point F.	throw S.	throw F.	layup S.	layup F.	2point S.	2point F.	dunk S.	dunk F.	steal	Mean
IDT [163]	0.370	0.501	0.778	0.365	0.283	0.278	0.136	0.303	0.197	0.004	0.555	0.343
IDT [163] player	0.428	0.481	0.703	0.623	0.300	0.311	0.233	0.285	0.171	**0.010**	0.473	0.365
C3D[157]	0.117	0.282	0.642	0.319	0.195	0.185	0.078	0.254	0.047	0.004	0.303	0.221
MIL[3]	0.237	0.335	0.597	0.318	0.257	0.247	0.224	0.299	0.112	0.005	0.843	0.316
LRCN[37]	0.462	0.564	0.876	0.584	0.463	0.386	0.257	0.378	0.285	0.027	0.876	0.469
Atten. no track[128]	0.583	0.668	0.892	0.671	0.489	0.426	0.281	0.442	0.210	0.006	0.886	0.505
Atten. track[128]	0.600	0.738	0.882	0.516	0.500	**0.445**	0.341	0.471	**0.291**	0.004	0.893	0.516
Ours	**0.753**	**0.766**	**0.933**	**0.857**	**0.613**	0.435	**0.405**	**0.542**	0.232	0.007	**0.940**	**0.589**

Table 3.1: Mean average precision for event classification given isolated clips of Basketball Dataset. "S." stands for "success" and "F." stands for "failure". All results except ours are extracted from [128].

The confusion matrix for all 11 actions is shown in Figure 3.2. By analyzing this confusion matrix, one can see that: **(1)** 18.09% "3-pointer success" test samples are incorrectly labeled as "2-pointer success" and 23.19% "3-pointer failure" are labeled as "2-pointer failure". In contrast, 12.16% and 16.86 % "2-pointer success/failure" test samples are incorrectly labeled as "3-pointer success/failure" correspondingly. Based on the rule specification "A player's feet must be completely behind the three-point line at the time of the shot or jump in order to make a three-point attempt; if the player's feet are on or in front of the line, it is a two-point attempt.[vii]", one can easily understand that sometime it is hard for a model (even for a person) to extract such detail

[vii] https://en.wikipedia.org/wiki/Three-point_field_goal

information to distinguish between 3-pointers and 2-pointers. Although the authors in [128] designed a model to locate the "shooter", they still cannot extract useful enough features to achieve a better performance than our proposed scheme. **(2)** 53.7% and 60.0% "slam dunk success/failure" are predicted as "layup success/failure". The reason is two-fold: a. the training data for "slam dunk success/failure" are not enough; b. "layup" and "slam dunk" have similar action patterns (the shooter jumps under the net and sends the ball to the net).

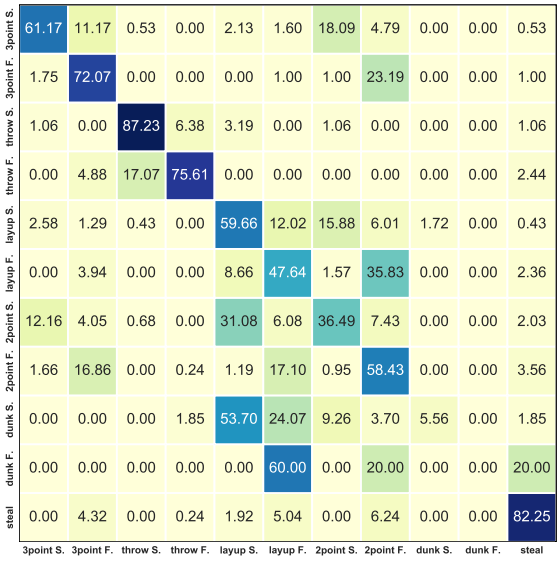


Figure 3.2: Confusion matrix of action recognition results on NCAA Basketball Dataset.

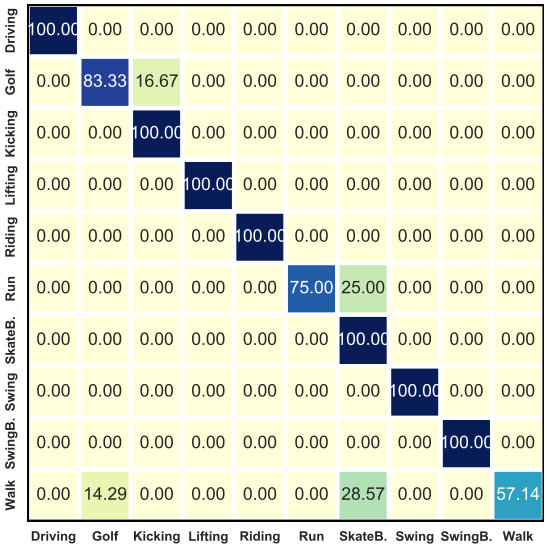


Figure 3.3: Confusion matrix of action recognition results on UCF Sports Action Dataset.

Here, we would like to highlight an interesting observation. If we group 10 shooting-related actions (except "steal") into two categories (success or failure), then we will get 717 success samples and 1122 failure samples in the testing subset. Based on the output of our model, 88% of the test samples (583 success and 1035 failure) are correctly labeled to these two categories. This observation proves that our scheme has the capability to distinguish between shooting success and shooting failure. Sometime, it is hard for people to judge if a shooting is success or not only based on the relative location between the ball and the net, let alone a designed model. Thus, we believe that our scheme achieves a good performance for it benefits from its capability to analyze players' behaviors before and after shooting, and infer the final activity label based on these behaviors. We will discuss more in Section 3.2.7 by visualizing our proposed model.

3.2.4 Experiments on the UCF Sports Action Dataset

In this subsection, we evaluate the performance of our scheme using the UCF Sports Action Dataset. We follow Lan et al. [86] to split the dataset into training (103 videos) and testing (47 videos) subsets[viii]. Among all video clips, the minimum length is 2.2 seconds (55 frames) and the maximum length is 14.4 seconds (360 frames). No matter how many frames in a video clip, we down-sampling all video clips to 24 frames before feeding them into our model. In Table 3.2, we compare our scheme to other state-of-the-art solutions. Our scheme gets the highest prediction accuracy on 8 out of 10 categories. Comparing to [60], our scheme achieves 6.1% accuracy improvement. One reason why our model performs better than other schemes is that most existing works first locate people in every frame of a video clip and then recognizes their activities. The performance of their models highly depends on their ability to localize people. If the model cannot correctly detect/locate people, the model cannot precisely recognize their activities. However, our proposed model (ReHAR) does not rely on the people detection scheme. Specifically, it is trained to extract useful features (including environmental information and motion information) from the entire video frames, and classify human activities merely based on these extracted features. In addition, we want to highlight that our scheme performs a perfect prediction (1.0 average precision) on 6 categories. This proves that our model generates more distinguishing features that benefit our model performs better than other existing methods in the task of activity recognition.

	Diving	Golf	Kicking	Lifting	Riding	Run	SkateB.	Swing	SwingB.	Walk	mAP
Gkioxari et al. [53]	0.758	0.693	0.546	0.991	0.896	0.549	0.298	0.887	0.745	0.447	0.681
Weinzaepfel et al. [167]	0.607	0.776	0.653	1.000	0.995	0.526	0.471	0.889	0.629	0.644	0.719
Peng et al. [125]	0.961	0.805	0.735	0.992	0.976	0.824	0.574	0.836	0.985	0.760	0.845
Hou et al. [60]	0.844	0.908	0.865	0.998	1.000	**0.837**	**0.687**	0.658	0.996	0.878	0.867
Ours	**1.000**	**0.955**	**1.000**	**1.000**	**1.000**	0.806	0.626	**1.000**	**1.000**	**0.888**	**0.928**

Table 3.2: Mean average precision for event classification given isolated clips of UCF Sports Action Dataset. All results except ours are extracted from [60].

Please refer to Figure 3.3 for more details about our results on the UCF Sports Dataset. One can see that our scheme performs very well on most categories. However, it incorrectly labels some "Walking" testing samples to "Golf" and "SkateBoarding". This is because these samples have some similiar features as samples in those incorrect categories. For example, in video "Walk-Front/006RF1-13902_70016.avi", there is a person walking on a golf course with a golf pole. The

[viii] http://cs.stanford.edu/~taranlan/other/train_test_split

environment is definitely related to golf and the motion of the golf pole looks like a person is swinging the pole in front of him. More details will be discussed later by visualizing the model.

Considering that the UCF Sports Action Dataset only consists of 150 videos in total, merely using one training/testing subset split as Lan et al. [86] cannot evaluate the real capability of a model. Thus, we do 5-fold cross-validation and report the experimental results in Table 3.3. Specifically, we evenly split all videos belonging to the same category into five subsets and run the training and validation process for five times. Each time we choose one subset as the validation set and train the model using the remaining four subsets.

	Diving	Golf	Kicking	Lifting	Riding	Run	SkateB.	Swing	SwingB.	Walk	mAP
Split1	1.000	1.000	1.000	1.000	1.000	0.571	0.226	1.000	1.000	0.667	0.846
Split2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Split3	1.000	1.000	1.000	1.000	1.000	1.000	0.563	1.000	1.000	0.583	0.915
Split4	1.000	1.000	0.833	1.000	1.000	1.000	0.700	1.000	1.000	0.333	0.887
Split5	1.000	0.833	1.000	1.000	1.000	1.000	0.134	1.000	1.000	0.750	0.872
AVG	1.000	0.967	0.967	1.000	1.000	0.914	0.525	1.000	1.000	0.667	0.904

Table 3.3: Mean average precision of ReHAR in 5-fold cross-validation on UCF Sports Action Dataset.

From Table 3.3, one can see that ReHAR achieves high accuracy in 5-fold cross-validation on this dataset. The performance of ReHAR on different categories is consistent with the results in Table 3.2. The model performs pretty well in most categories but the "SkateBoard" and "Walking". Thus, we looked over all the videos of these two categories. Compared to other categories, videos of "SkateBoard" and "Walking" in this dataset have more variety of viewpoints and object sizes. Some videos are recorded with looking up angles while others are recorded from ordinary angles. Besides, some videos belonging to these two categories are shot from a great distance while others are from a close distance. These variations caused the model to perform worse on these two categories. To fix this problem, we need to include more training data with varying viewpoints and shooting distances, so that the model can be robust to these variations.

3.2.5 Comparison between SBGAR and ReHAR

As we discussed at the beginning of this chapter, ReHAR is designed to remove some limitations of SBGAR. Here, we compare the performance of SBGAR and ReHAR using the Volleyball Dataset. Table 3.4 shows the details.

Methods	Accuracy (%)
S1 - SBGAR (RGB Frame Only)	38.7
S2 - SBGAR (Optical Flow Only)	54.3
S3 - SBGAR (RGB & Optical Flow)	66.9
R1 - ReHAR (RGB Frame Only)	50.6
R2 - ReHAR (Optical Flow Only)	69.6
R3 - ReHAR (RGB & Optical Flow)	70.7

Table 3.4: Comparison of ReHAR with SBGAR on the Volleyball Dataset.

From Table 3.4, we can see that ReHAR performs better than SBGAR on the Volleyball Dataset. If we compare S1 with R1 (or compare S2 to R2), we can see that ReHAR achieves much higher accuracy (more than 10% accuracy improvement) than SBGAR. It proves that the proposed end-to-end trainable structure indeed help ReHAR extract more useful features from its inputs. It is interesting that R3 only performs 1% better than R2. We believe that it is caused by the particularity of the Volleyball dataset. All videos in this dataset are volleyball game-related, so they have a similar background. In such a dataset, the motion clue is more important than the environmental information for the human activity recognition task. Thus, for this dataset, ReHAR extracts motion features from optical flow images which is sufficient for the human activity recognition task, and hence adding RGB frames does not further improve the performance much. Even so, we still suggest using RGB and Optical Flow images simultaneously to handle variations of different datasets.

Although ReHAR performs a little better than SBGAR on the Volleyball dataset, we want to argue that both ReHAR and SBGAR are useful and necessary for different application scenarios. On the one hand, during training, ReHAR only requires video classification labels as ground truth. It is helpful for the scenarios that accuracy is more important and only video level category labels are accessible. On the other hand, SBGAR generates dense semantic representation. Such a characteristic is more useful for video retrieval, video description, story generation, and so on.

3.2.6 Computation Time

As we have already discussed before, in some application scenarios, predicting an activity label in real time is highly important. Thus, in this subsection, we report the computation time of ReHAR. Computing optical flow images takes FlowNet 2.0 [66] around 7ms (140 fps). We report the

computation time of ReHAR (including optical flow images generation time) using different CNN models as base net in Table 3.5. In total, our model (using VGG16 as its base net) takes 103.65 ms to process 10 input frames and 239.04 ms for 24 input frames. In [94], the computation time of SBGAR model using InceptionV3 as feature extractor and 10 input frames was 108.53 ms. Using the same settings, ReHAR only takes 78.40 ms. Considering that both [65] and [128] predict the activities based on detecting and analyzing every single person and then infer the final activities based on individual actions, they have a similar computation time (4.2 seconds on a GTX 1080 reported in [94]). ReHAR runs an order of magnitude faster than [65] and [128]. Thus, our scheme will be more useful for real-time human activity recognition.

CNN base net	Time on 10 Frames (ms)	Time on 24 Frames (ms)
VGG16	103.65	239.04
InceptionV3	78.40	192.02

Table 3.5: Computation time of ReHAR using different CNN model as its base net (optical flow images generation time included).

3.2.7 Why does our scheme work?

In previous subsections, we report our comparable results on two well-known activity recognition datasets. In this subsection, we will try to explain the reason why our proposed model works.

First, we explore the necessity of the LSTM1 and the Global Pooling layers in our scheme by comparing baselines' results on UCFSports dataset. Our proposed model achieves 0.928 mAP (Table 3.2). **(1)** If we remove LSTM1, stack and feed the output of global layers to a Convolutional layer before "FC1" layer, the mAP reduces to 0.766. **(2)** We only get 0.702 mAP after replacing the LSTM1 with an element-wise sum operation. Baseline (1) and (2) are the best fusion methods discussed in [44]. One can see that our LSTM1 generates much better representations than a simple fusion method. **(3)** Replacing the Global Pooling layers with flattened layers, the mAP reduces from 0.928 to 0.889. Thus, using Global Pooling layers helps our model achieve a higher accuracy.

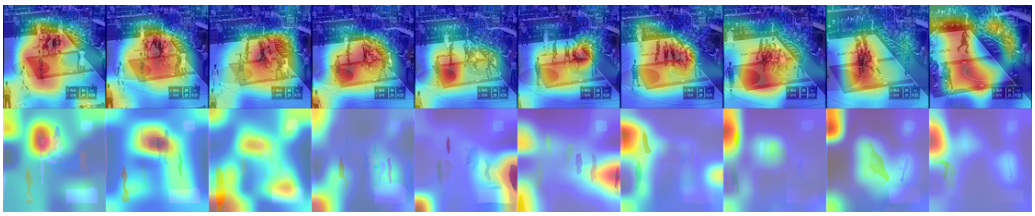
Then, we use the method proposed in [139] to compute the gradient of output category with respect to input image. This should tell us how the output category value changes with respect to a small change in input image pixels. We implement this function by modifying the keras-vis toolkit [80], so that the final class-specific information can be passed back through two LSTMs and

fully-connected layers.

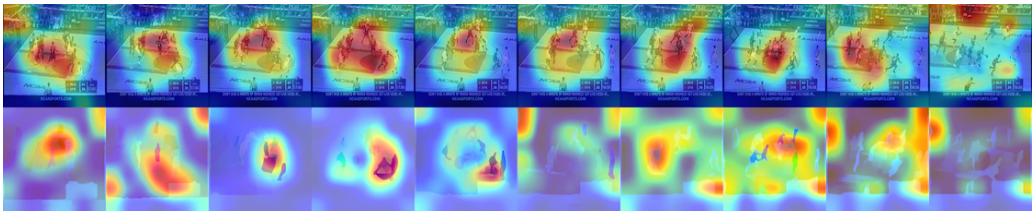
Figure 3.4 shows the visualized results using four samples from the Basketball Dataset and the UCF Sports Action Dataset. **Figure 3.4a** shows an "other 2-pointer success" event and our model correctly predicts it. One can notice that it is hard for a person to find and track the ball through all frames. The visualized result shows that our model focuses on analyzing the players instead of tracking the ball. At the first frame, the group of the players on the video frame draws the attention of our model, while the model pays more attention on the region of the shooter on the optical flow image. At the last frame, the model stares at the region under the net with only one player left. Based on these information as well as features extracted from intermediate frames, the model predicts a correct activity label. **Figure 3.4b** shows one player successfully steals the ball from another at sixth frame, and then all players are running towards the other side of the basketball court. The model focuses on a larger region of optical flow images after the sixth frame, because all players (including environment) are moving quickly. From **Figure 3.4c**, one can notice that the model has the capability to detect the key actor from video frames. There are two people on the images and the model highlights the shooter rather than the referee on most frames. In addition, the model highlights the location of the ball at the last 4 optical flow images. All of these prove that our designed model can focus on important and meaningful things. In **Figure 3.4d**, we visualize a sample that our model wrongly predicts a "Walking" event to "Golf". The visualized result shows that the model extracts features from the person and the background context on video frames. These features contributes toward "Golf" event. We also notice that at the last optical flow image, the model highlights the region of the golf pole which is located between the person's two legs. Maybe these are the reasons why the model has 97% confidence to label this sample as "Golf".

3.3 Discussion

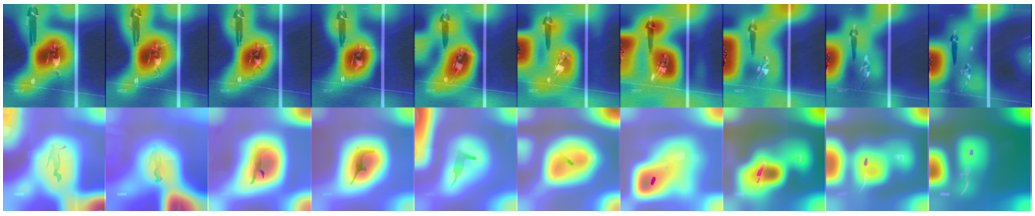
In this chapter, we propose a novel scheme (ReHAR) to recognize human activities in videos. The proposed model is trainable end-to-end and achieves a higher accuracy than the existing state-of-the-art solutions on both single person activity and group activity datasets. The experimental results also show that ReHAR runs an order of magnitude faster than other schemes. By visualizing the



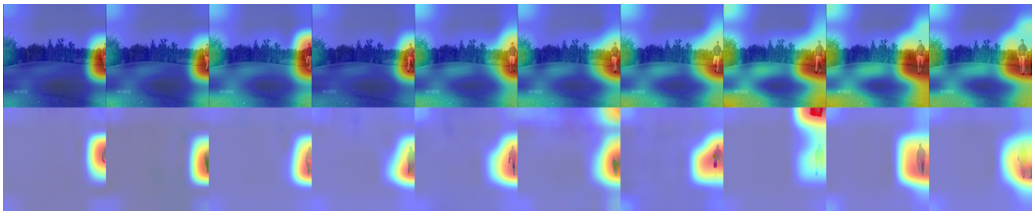
(a) Correctly predict an “other 2-pointer success” event on Basketball Dataset.



(b) Correctly predict a “Steal Success” event on Basketball Dataset.



(c) Correctly predict a “Kicking” event on UCF Sports Action Dataset.



(d) Incorrectly predict a “Walking” event as “Golf” on UCF Sports Action Dataset.

Figure 3.4: Visualized class-specific important regions on input video frames and optical flow images. The top 2 samples are from Basketball Dataset and the bottom 2 are from UCF Sports Dataset. Each sample has two rows of visualized results. The first row shows the results of video frames, while the second row illustrates the results of optical flow images. Because of the limitation of the space, we only visualize 10 frames of each event.

proposed model, we understand what ReHAR learns and notice that it has the potential capability to detect key actors.

First, similar to SBGAR, ReHAR generates an independent representation for each input video frame thus it is also robust to the situation that the model wrongly generates an intermediate representation at a time step. Such a strategy helps our models achieve higher prediction accuracy than existing methods. Second, compared to SBGAR (proposed in the previous chapter), ReHAR does not require datasets with semantic caption annotations, which makes the ReHAR scheme

more general for different application scenarios. Third, different from SBGAR, ReHAR generates the intermediate representation with a shorter and fix LSTM iterations (LSTM1 only requires 4-time steps for 4 types of global features as inputs and 1 output as shown in Figure 3.1). However, SBGAR needs to generate a longer semantic caption (generates more than 4 words which results in more LSTM iterations) for each frame. Thus, ReHAR runs faster than SBGAR. Fourth, ReHAR is end-to-end trainable thus all layers are trained for the final prediction. The visualized class-specific important regions (shown in Figure 3.4) prove that ReHAR focuses on the human body motion parts. Features extracted from these motion parts are definitely useful for the human activity recognition task.

Chapter 4

GRIP: Graph-based Interaction-aware Trajectory Prediction

In the past two chapters, we proposed and improved schemes for human activity recognition. The proposed models perform better than the existing state-of-the-art solutions. However, we notice that there are some limitations when we apply a human activity recognition model in real life. The biggest weakness of a human activity recognition scheme is that the model only works pretty well after an activity has finished. Such a limitation results in an activity recognition scheme becomes useless in some specific application scenario, e.g., smart video surveillance and warning system or self-driving cars. In these scenarios, the control systems require a capability of not only detecting and tracking objects but also predicting their motion intents quickly, so that the system can proactively react toward these intents for safety purposes. Most existing work [113, 21, 21, 124, 54, 110, 35], as we discussed in Chapter 1, either perform badly in predicting trajectories of objects because they ignore the impacts of their nearby objects, or run slowly due to they only anticipate the future location of one particular object.

Thus, in this chapter, we propose a robust and efficient object trajectory prediction scheme [96]. The proposed solution is a general model that can be used for smart video surveillance systems, autonomous driving cars, or other trajectory prediction related scenarios. In Section 4.1, we describe the problem formulation followed by the explanation of our proposed object trajectory prediction scheme and implementation details in Section 4.2. We report our experimental results

in Section 4.3. Finally, we conclude this chapter in Section 4.4.

4.1 Problem Formulation

Before introducing our proposed scheme, we would like to formulate the trajectory prediction problem as estimating the future positions of all objects in a scene based on their trajectory histories. The objects can be joints of a human skeleton, traffic agents perceived by an autonomous driving car, or other objects whose trajectory can be tracked. Specifically, the inputs X of our model are trajectory histories (over t_h time steps) of all observed objects:

$$X = [p^{(1)}, p^{(2)} \cdots, p^{(t_h)}] \tag{4.1}$$

where,

$$p^{(t)} = [x_0^{(t)}, y_0^{(t)}, x_1^{(t)}, y_1^{(t)}, \cdots, x_n^{(t)}, y_n^{(t)}] \tag{4.2}$$

are the co-ordinates of all observed objects at time t, and n is the number of observed objects. This format is the same as what Deo et al. defined in [36] and [35].

Considering that we feed track histories of all observed objects into the model, we argue that it makes more sense to predict future positions for all of them simultaneously. Thus, instead of only predicting the position of one particular object as done in [36] and [35], the outputs Y of our proposed model are the predicted positions of all observed objects from time step $t_h + 1$ to $t_h + t_f$ in the future:

$$Y = [p^{(t_h+1)}, p^{(t_h+2)}, \cdots, p^{(t_h+t_f)}] \tag{4.3}$$

where $p^{(t)}$ is the same as equation (4.2) and t_f is the predicted horizon.

4.2 Proposed Scheme

To solve the limitations of existing approaches, we propose a novel deep learning model for object trajectory prediction in this section. Our model, illustrated in Figure 4.1, consists of three components: (1) Input Preprocessing Model, (2) Graph Convolutional Model, and (3) Trajectory Prediction Model.

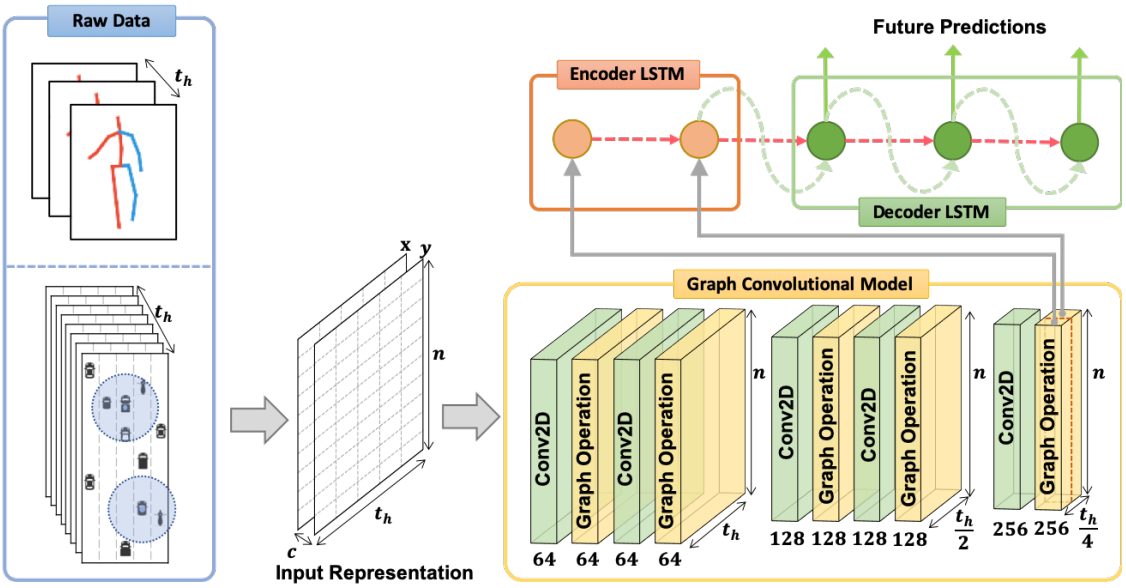


Figure 4.1: The architecture of the proposed Scheme.

4.2.1 Input Preprocessing Model

1) Input Representation

Before feeding the trajectory data of objects into our model, we convert the raw data into a specific format for subsequent efficient computation. Assuming that n joints of a human body (or n objects in a traffic scene) were observed in the past t_h time steps, we represent such information in a 3D array F_{input} with a size of $(n \times t_h \times c)$ (as shown in Figure 4.1). Each row (n, the first dimension) corresponds to one object, each column (t_h, the second dimension) consists of the status of all objects at that time step, and c is the coordinates of locations. In this thesis, we set c accordingly to indicate coordinates (x, y or more dimensions) of an object. For example, $c = 2$ in the self-driving application scenario, and $c = 3$ for a 3D coordinates of human body joints. All coordinates are normalized to the range of $(-1, 1)$. Thus, the trajectory data is represented as an image (with a size of $w*h*c$), so we can use techniques that work pretty well on images to process trajectory data.

2) Graph Construction

Considering that, in the human motion prediction application scenario, the motion of a joint is impacted by its connected joints (similarly, in the autonomous driving application scenario, the motion of an object is profoundly impacted by the movements of its surrounding objects). This is highly similar to people's behaviors on a social network (one person is usually to be impacted by his/her friends). This inspires us to represent the inter-object interaction using an undirected graph $G = \{V, E\}$ as what researchers have done for a social network.

In this graph, each node in node set V corresponds to a joint of a human body (or an object in a traffic scene). Considering that each object may have different states at different time steps, the node set V is defined as $V = \{v_{it} | i = 1, \cdots, n, t = 1, \cdots, t_h\}$, where n is the number of considered joints on a human body (or the number of observed objects in a scene), and t_h is the observed time steps. The feature vector v_{it} on a node is the coordinate of ith object at time t.

At each time step t, objects that have interactions should be connected with edges. In the human motion prediction application scenario, joints are connected according to the human skeleton. Thus, joints connection is fixed. However, in the autonomous driving application scenario, such an interaction happens when two objects are close to each other. The edge set E is composed of two parts: (1) The first part describes the interaction information between two objects in spatial space at time t. We call it "spatial edge" and denote it as $E_S = \{v_{it}v_{jt} | (i, j \in D)\}$, where D is a set in which objects are close to each other. In this dissertation, we define that two objects are close if their distance is less than a threshold of D_{close}. In Figure 4.1, we demonstrate this concept on "Raw Data" using two blue circles with a radius of D_{close}. All objects within the blue circle are regarded as close to the one located in the middle of the circle. Thus, the top object has three close neighbors, and the lower one only has one neighbor. (2) The second part is the inter-frame edges, which represents the historical information frame by frame in temporal space. Each observed object in one time-step is connected to itself in another time-step via the temporal edge and such edges are denoted as $E_F = \{v_{it}v_{i(t+1)}\}$. Thus, all edges in E_F of one particular object represent its trajectory over time steps.

To make the computation more efficient, we represent this graph using an adjacency matrix

$A = \{A_0, A_1\}$, where A_0 is an identity matrix I representing self-connections in temporal space, and A_1 is a spatial connection adjacency matrix. Thus, at any time t,

$$A_0[i][j](or A_1[i][j]) = \begin{cases} 1, \text{if edge } \langle v_{it}, v_{jt} \rangle \in E \\ 0, \text{otherwise} \end{cases} \tag{4.4}$$

Both A_0 and A_1 have a size of $(n \times n)$, where n equals to the number of considered joints of a human body (or the number of observed traffic objects in a scene).

4.2.2 Graph Convolutional Model

The Graph Convolutional Model consists of several convolutional layers as well as graph operations. These convolutional layers are designed to capture useful temporal features, e.g., motion pattern of one object, and graph operations to handle the inter-object interaction in spatial space. Thus, as shown in Figure 4.1 (5 convolutional layers and 5 graph operation layers are illustrated), one graph operation layer is added to the end of each convolutional layer in this Graph Convolutional Model to process the input data temporally and spatially alternatively. Based on the common experience, a deeper layer of a CNN network extracts higher-level semantic meaning features than a lower layer. Besides, each channel of a CNN layer represents one type of features. Thus, we increase the number of channels (from 64 to 256) of a layer as the depth goes deeper.

1) Convolutional Layer

Given a preprocessed input data $F_{input} := \mathbb{R}^{N \times T \times C}$ (where N is the number of objects, T is the observed time steps, and C is the dimension of the coordinates), the model first passes it through a convolutional layer to compute convolutional feature maps f_{conv}. We set the kernel size of convolutional layers to (1×3) to force them to process the data along the temporal dimension (second dimension). Appropriate paddings and strides are added to make sure that each layer has an output feature map with expected size.

2) Graph Operation Layer

Then, we feed the generated convolutional feature maps f_{conv} to a graph operation layer to take the interactions of connected joints (or surrounding objects) into account. The graph operation involves multiplying normalized version of matrix A with f_{conv} using the following formula:

$$f_{graph} = \sum_{j=0}^{1} \Lambda_j^{-\frac{1}{2}} A_j \Lambda_j^{-\frac{1}{2}} f_{conv} \tag{4.5}$$

where A is the adjacency matrix we constructed in subsection 4.2.1 and Λ_j is computed as:

$$\Lambda_j^{ii} = \sum_k (A_j^{ik}) + \alpha \tag{4.6}$$

$\Lambda^{-\frac{1}{2}} A \Lambda^{-\frac{1}{2}}$ is a normalized version of A, which is used to make sure that the value range of feature maps remain unchanged after performing the graph operations.We set $\alpha = 0.001$ to avoid empty rows in A_j.

In Figure 4.2, we explain the effect of our Graph Convolutional model using two examples. The upper row shows an example using human joints data while the lower row is an autonomous-driving example. Let's take one human joint (shoulder) as an example. Before feeding an input data (Figure 4.2(a)) into a Graph Convolutional Neural (GCN) model, one joint (marked using a red circle) only consists of its own information (coordinates). After we pass the data through one GCN layer (Figure 4.2(b)), the information of the red joint is updated based on all highlighted joints (circled with a red dashed line). These highlighted joints are connected to the red joint, so the GCN considers the impact of all of them. If we pass the data through more GCN layers (Figure 4.2 (c) and (d)), more related joints will be considered. Thus, each layer considers joints within different ranges, and after a few layers, all joints will be included. In other words, our model extracts feature based on human structural information and gradually expand its receptive field.

It is similar for the autonomous driving example. The only difference is that the connected traffic agents are not fixed and pre-designed, but are decided based on distances. Each time, one GCN layer expands its receptive field by considering all nearby objects of highlighted traffic agents (within a red-dashed circle).

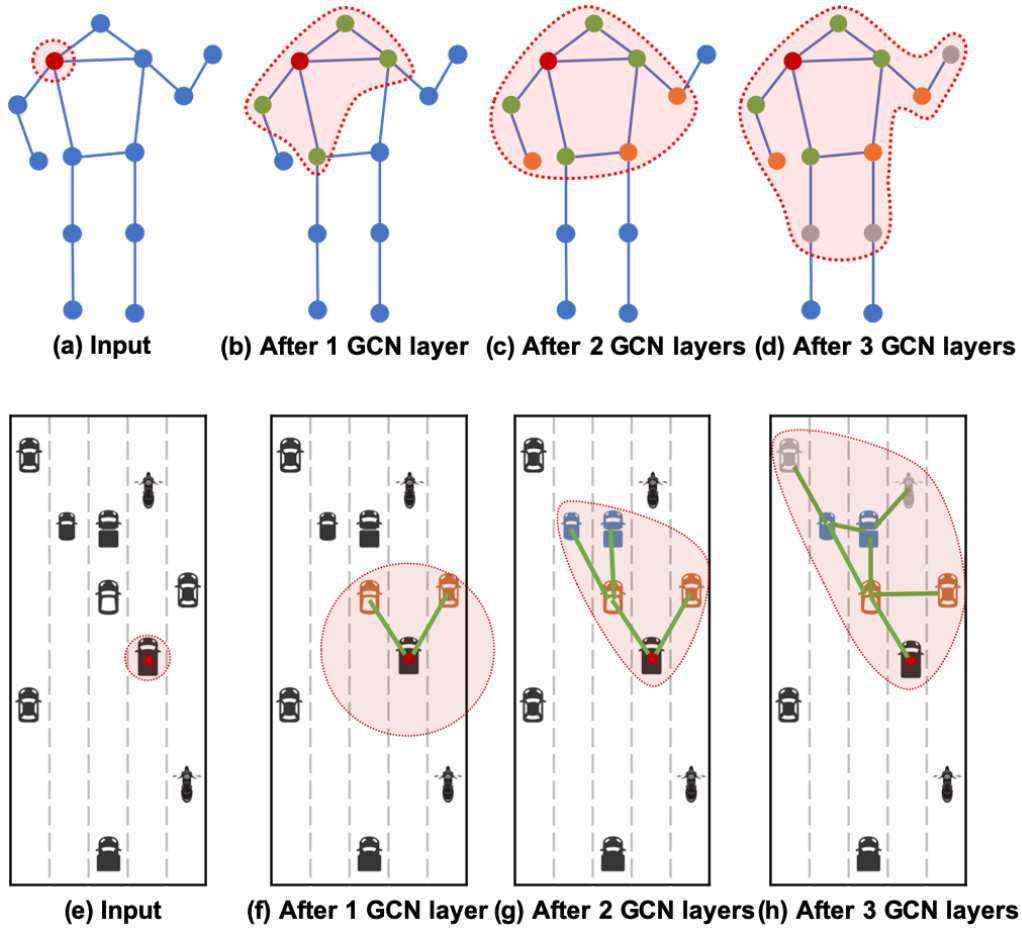


Figure 4.2: The effect of a Graph Convolutional model.

4.2.3 Trajectory Prediction Model

This model predicts the future trajectories for all considered joints (or observed objects in a scene) simultaneously. The Trajectory Prediction Model is an LSTM encoder-decoder network that takes the computed output of the Graph Convolutional Model f_{graph} as input. The output of the graph convolutional model is fed into the encoder LSTM at each time step. Then, the hidden feature of the encoder LSTM, as well as coordinates of objects at the previous time step, are fed into a decoder LSTM to predict the position coordinates at the current time step. Such a decoding process is repeated several times until the model predicts positions for all expected time steps (t_f) in the future.

4.2.4 Implementation Details

Our scheme is implemented using Python Programming Language and PyTorch Library [121]. We report the implementation details of our scheme and the settings of important parameters as follows.

Input Preprocessing Model: For human motion prediction, we consider n joints (n is variant for different datasets) of a human body. n joints are connected according to the human skeleton. For self-driving application scenario, we process a traffic scene within 180 feet ($\pm$ 90 feet). All objects within this region will be observed and predicted in the future. While constructing the graph, we consider two objects are close if their distance is less than 25 feet ($D_{close} = 25$). Thus, any pair of objects within 25 feet are connected using a spatial edge, $e_s \in E_S$. Please refer to our ablation study in section 4.3.5 for more details.

Graph Convolutional Model: The Graph Convolutional Model consists of 10 convolutional layers, denoted as $\{conv2d\_i | i = 1, 2, \cdots, 10\}$. All Conv2D layers have a convolutional kernel with a size of (1×3). Among all of these 10 Conv2D layers, we set $stride = 2$ for $conv2d\_5$ and $conv2d\_8$ to achieve some pooling effects, but use $stride = 1$ for remaining layers. The output channel of the first Conv2D is set to 64. We double the number of output channels when $stride = 2$. Thus, the final output of the Graph Convolution Model has 256 channels.

Each of these convolutional layers is followed by a graph operation layer. Graph operation layers do not change the size of features, and they share the same adjacency matrix. To avoid overfitting, we randomly dropout features (0.5 probability) after each graph operation.

Trajectory Prediction Model: Both the encoder and decoder of this prediction model are a two-layer LSTM. We set the number of hidden units of these two LSTMs equals to the output dimension (e.g., $2 \times n$, where n is the number of objects and 2 is the x, y coordinates). The input of the encoder has 256 channels that are the same as the output of the Graph Convolutional Model. We add a $tanh$ activation function to the output layers of both LSTMs to rescale the output to range of (-1, 1).

Optimization: We train our model as a regression task at each time. The overall loss can be

computed as:

$$Loss = \frac{1}{t_f} \sum_{t=1}^{t_f} loss^t \tag{4.7}$$

$$= \frac{1}{t_f} \sum_{t=1}^{t_f} \left\| Y_{pred}^t - Y_{GT}^t \right\|^2 \tag{4.8}$$

where t_f is the time step in the future (in Figure 4.1, $t_f = 3$), $loss^t$ is the loss at time t, Y_{pred} and Y_{GT} are predicted positions and ground truth respectively. The model is trained to minimize the $Loss$.

Training Process: We train the model using Stochastic Gradient Descent (SGD) optimizer with 0.001 starting learning rate. The learning rate is reduced by multiplying with 0.1 once per 5 epochs until the loss becomes converged. As done in [35], we set $batch\_size = 128$ during training.

4.3 Experiments

We evaluate our proposed model in two tasks (1) human motion prediction and (2) traffic agents trajectory prediction. The model was run on a desktop running Ubuntu 16.04 with 4.0 GHz Intel Core i7 CPU, 32GB Memory, and a NVIDIA Titan Xp Graphics Card.

4.3.1 Datasets

Human 3.6M Dataset: The Human 3.6M dataset [67] is one of the largest datasets of human motion capture dataset. This dataset consists of motion capture data from 7 actors performing 15 actions. In this dataset, 32 joint locations of each person are provided. Following previous work [69, 113, 21], we use Subject 5 as the test data and Subjects 1, 6, 7, 8, 9, 11 as training. Consistent with the previous work [69, 113, 21], we train our model using the past 50 frames (2000 s) and predict the future 10 frames (400 ms) for short-term prediction. For the long-term forecast, the model predicts the next 25 frames (2000 ms).

Penn Action Dataset: The Penn Action Dataset [183] provides 13 human joint coordinates over 15 different actions. In total, this dataset consists of 2326 video sequences. Following [183,

15, 21], we split the dataset into 1258 training video sequences and 1068 testing video sequences. For this dataset, the model only takes the first frame as its input, and predicts the next 15 frames.

NGSIM Dataset: We evaluate our scheme on two well known traffic trajectory prediction datasets: NGSIM I-80 [27] and US-101 [26]. Both datasets were captured at 10 Hz over 45 minutes and segmented into 15 minutes of mild, moderate and congested traffic conditions. These two datasets consist of trajectories of vehicles on real freeway traffic. Coordinates of cars in a local coordinate system are provided. We follow Deo et al. [36, 34, 35] to split these two datasets into training and testing sets. One-fourth of the data from each of the three subsets (mild, moderate, and congested traffic conditions) are selected for testing. Each trajectory is segmented into 8 seconds clips that the first 3 seconds are used as observed track history and the remaining 5 seconds are the prediction ground truth. To make a fair comparison, we also do the same downsampling for each segment by a factor 2 as Deo et al. did, i.e. 5 frames per second. The code for dataset segmentation can be downloaded from their Github [i].

4.3.2 Metrics

Three metrics are used in this chapter:

MAE distance: Following [21], mean average error (MAE) is used for the Human 3.6M dataset. It measures the mean average distance between the predicted pose in the angle space and the ground-truth pose.

PCK@0.05: The PCK@0.05 metric is used for the Penn Action dataset as done in [15, 21]. The PCK metric calculates the percentage of joint locations correctly predicted by the model. With the threshold 0.05, a joint location is counted as correctly predicted if the normalized distance between its predicted and ground-truth locations is less than 0.05. The distance is normalized typically based on the size of the full body or the head. Since we have the coordinates of human body joints, we normalize the distance by $max(h, w)$, where h and w are the height and width of the human body (longest distance between any two joints in height and width dimensions separately).

RMSE: We use the same experimental settings and evaluation metrics as [35] and [82] for NGSIM datasets. In this chapter, we report our results in terms of the root of the mean squared

[i] https://github.com/nachiket92/conv-social-pooling

error (RMSE) of the predicted trajectories in the future (5 seconds horizons). The RMSE at time t can be computed as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_{pred}^{t}[i] - Y_{GT}^{t}[i])^2} \tag{4.9}$$

where n is the number of observed (predicted) objects, Y_{pred}^{t} and Y_{GT}^{t} are predicted results and ground truth at time t correspondingly.

4.3.3 Experimental Results on Human 3.6M Dataset

We first evaluate the performance of our proposed GRIP in the filed of human motion prediction using the Human 3.6M dataset. In Table 4.1, we report short-term prediction results over 4 common actions from Human 3.6M dataset.

From this table, one can see that our proposed GRIP achieves better results than the state-of-the-art solution. GRIP only performs a little worse (0.01 worse in terms of MAE) than TP-RNN at 320 ms for the “Eating" action. Besides that, GRIP performs better than all existing solutions. Especially for “Smoking” and “Discussion”, GRIP achieves much lower error than the state-of-the-art scheme (QuaterNet). These results prove that GRIP performs well in the human motion short-term prediction task.

	Walking				**Eating**				**Smoking**				**Discussion**			
milliseconds	**80**	**160**	**320**	**400**	**80**	**160**	**320**	**400**	**80**	**160**	**320**	**400**	**80**	**160**	**320**	**400**
ERD [49]	0.93	1.18	1.59	1.78	1.27	1.45	1.66	1.80	1.66	1.95	2.35	2.42	2.27	2.47	2.68	2.76
LSTM-3LR [49]	0.77	1.00	1.29	1.47	0.89	1.09	1.35	1.46	1.34	1.65	2.04	2.16	1.88	2.12	2.25	2.23
SRNN [69]	0.81	0.94	1.16	1.30	0.97	1.14	1.35	1.46	1.45	1.68	1.94	2.08	1.22	1.49	1.83	1.93
Residual [113]	0.28	0.49	0.72	0.81	0.23	0.39	0.62	0.76	0.33	0.61	1.05	1.15	0.31	0.68	1.01	1.09
TP-RNN [21]	0.25	0.41	0.58	0.65	0.20	0.33	**0.53**	0.67	0.26	0.47	0.88	0.90	0.30	0.66	0.96	1.04
QuaterNet [124]	0.21	**0.34**	0.56	**0.62**	0.20	0.35	0.58	0.70	0.25	0.47	0.93	0.90	0.26	0.60	0.85	0.93
GRIP (Ours)	**0.20**	**0.34**	**0.55**	**0.62**	**0.17**	**0.30**	0.54	**0.66**	**0.23**	**0.43**	**0.85**	**0.82**	**0.23**	**0.57**	**0.82**	**0.90**

Table 4.1: MAE for short-term prediction over four actions from Human 3.6M Dataset. In each column, the best results are typeset in boldface and the second best are underlined (the lower the better).

We also report GRIP’s long-term prediction results in Table 4.2 and Table 4.3. From Table 4.2, one can see that GRIP achieves better results at all considered future time steps on 4 common actions of the Human 3.6M dataset. If we compare Table 4.1 and Table 4.2, we can notice that GRIP performs better in making long-term predictions than short-term predictions. Specifically, when

making long-term predictions, GRIP gains more improvement from the stat-of-the-art solution than making short-term predictions. It is because that GRIP considers the impacts of nearby objects, which helps the model make a longer precise prediction.

	Walking					**Eating**					**Smoking**					**Discussion**				
milliseconds	**80**	**160**	**320**	**560**	**1000**	**80**	**160**	**320**	**560**	**1000**	**80**	**160**	**320**	**560**	**1000**	**80**	**160**	**320**	**560**	**1000**
ERD [49]	1.30	1.56	1.84	2.00	2.38	1.66	1.93	2.88	2.36	2.41	2.34	2.74	3.73	3.68	3.82	2.67	2.97	3.23	3.47	2.92
LSTM-3LR [49]	1.18	1.50	1.67	1.81	2.20	1.36	1.79	2.29	2.49	2.82	2.05	2.34	3.10	3.24	3.42	2.25	2.33	2.45	2.48	2.93
SRNN [69]	1.08	1.34	1.60	1.90	2.13	1.35	1.71	2.12	2.28	2.58	1.90	2.30	2.90	3.21	3.23	1.67	2.03	2.20	2.39	2.43
Droupoout-AE [52]	1.00	1.11	1.39	1.55	1.39	1.31	1.49	1.86	1.76	2.01	0.92	1.03	1.15	1.38	1.77	1.11	1.20	1.38	1.53	1.73
Residual [113]	0.32	0.54	0.72	0.86	0.96	0.25	0.42	0.64	0.94	1.30	0.33	0.60	1.01	1.23	1.83	0.34	0.74	1.04	1.43	1.75
TP-RNN [21]	0.25	0.41	0.58	0.74	0.77	0.20	0.33	**0.53**	0.84	1.14	0.26	0.48	0.88	0.98	1.66	0.30	0.66	0.98	1.39	1.74
GRIP (Ours)	**0.22**	**0.37**	**0.56**	**0.69**	**0.73**	**0.19**	**0.32**	**0.53**	**0.66**	**1.13**	**0.24**	**0.45**	**0.87**	**0.94**	**1.58**	**0.25**	**0.58**	**0.84**	**1.29**	**1.67**

Table 4.2: MAE for long-term prediction over four actions from Human 3.6M Dataset. In each column, the best results are typeset in boldface and the second best are underlined (the lower the better).

In Table 4.3, we report the remaining 11 actions in Human 3.6M dataset and the average prediction results over all of the actions. GRIP achieves better long-term predictions for most actions. Precisely, GRIP only performs worse at 7 predictions out of 66 predictions than the existing solutions. In average ("Average of all 15" columns), GRIP improves the performance of the state-of-the-art solution by almost 0.1 in terms of MAE.

	Directions						**Greeting**						**Talking on the phone**					
milliseconds	80	160	320	400	560	1000	80	160	320	400	560	1000	80	160	320	400	560	1000
Residual [113]	0.44	0.69	0.83	0.94	1.03	1.49	0.53	0.88	1.29	1.45	1.72	1.89	0.61	1.12	1.57	1.74	1.59	1.92
TP-RNN [21]	0.38	0.59	0.75	0.83	0.95	1.38	0.51	0.86	1.27	1.44	1.72	1.81	0.57	1.08	1.44	1.59	**1.47**	**1.68**
GRIP (Ours)	**0.34**	**0.49**	**0.70**	**0.80**	**0.89**	**1.28**	**0.45**	**0.75**	**1.11**	**1.27**	**1.53**	**1.62**	**0.56**	**1.07**	**1.41**	**1.55**	1.52	1.72
	Posing						**Purchases**						**Sitting**					
milliseconds	80	160	320	400	560	1000	80	160	320	400	560	1000	80	160	320	400	560	1000
Residual [113]	0.47	0.87	1.49	1.76	1.96	**2.35**	0.60	0.86	1.24	1.30	1.58	**2.26**	0.44	0.74	1.19	1.40	1.57	2.03
TP-RNN [21]	0.42	0.76	1.29	1.54	1.75	2.47	0.59	**0.82**	**1.12**	**1.18**	**1.52**	2.28	0.41	0.66	1.07	1.22	1.35	1.74
GRIP (Ours)	**0.22**	**0.49**	**1.09**	**1.34**	**1.61**	**2.35**	**0.58**	0.83	1.18	1.23	1.55	2.31	**0.34**	**0.53**	**0.91**	**1.09**	**1.22**	**1.58**
	Sitting down						**Taking photo**						**Waiting**					
milliseconds	80	160	320	400	560	1000	80	160	320	400	560	1000	80	160	320	400	560	1000
Residual [113]	0.51	0.93	1.44	1.65	1.94	2.55	0.33	0.65	0.97	1.09	1.19	1.47	0.34	0.65	1.09	1.28	1.61	2.27
TP-RNN [21]	0.41	0.79	1.13	1.27	1.47	1.93	0.26	0.51	0.80	0.95	1.08	1.35	0.30	0.60	1.09	1.31	1.71	2.46
GRIP (Ours)	**0.35**	**0.68**	**0.99**	**1.11**	**1.28**	**1.84**	**0.22**	**0.45**	**0.72**	**0.85**	**0.97**	**1.18**	**0.28**	**0.55**	**0.99**	**1.20**	**1.54**	**2.23**
	Walking dog						**Walking together**						**Average of all 15**					
milliseconds	80	160	320	400	560	1000	80	160	320	400	560	1000	80	160	320	400	560	1000
Residual [113]	0.56	0.95	1.28	1.39	1.68	1.92	0.31	0.61	0.84	0.89	1.00	1.43	0.43	0.75	1.11	1.24	1.42	1.83
TP-RNN [21]	0.53	0.93	1.24	1.38	1.73	1.98	0.23	0.47	0.67	0.71	0.78	1.28	0.37	0.66	0.99	1.11	1.30	1.71
GRIP (Ours)	**0.50**	**0.84**	**1.14**	**1.27**	**1.56**	**1.87**	**0.21**	**0.43**	**0.59**	**0.63**	**0.69**	**1.24**	**0.33**	**0.59**	**0.91**	**1.02**	**1.20**	**1.62**

Table 4.3: MAE for long-term prediction over the remaining 11 actions in Human 3.6M Dataset. In each column, the best results are typeset in boldface (the lower the better).

4.3.4 Experimental Results on Penn Action Dataset

We evaluate the performance of our GRIP on another human motion prediction dataset, the Penn Action dataset. In Table 4.4, we compare our GRIP with some prior works in terms of PCK@0.05. The values in Table 4.4 are the higher, the better. On this dataset, GRIP also achieves better than the state-of-the-art solutions at most time steps (14 out of 16 predictions). Consistently, GRIP works much better in making long-term predictions than other schemes.

Future Frame	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Residual [113]	82.4	68.3	58.5	50.9	44.7	40.0	36.4	33.4	31.3	29.5	28.3	27.3	26.4	25.7	25.0	24.5
3D-PFNet [15]	79.2	60.0	49.0	43.9	41.5	40.3	39.8	39.7	40.1	40.5	41.1	41.6	42.3	42.9	43.2	43.3
TP-RNN w/o init vel. [21]	82.3	68.9	61.5	56.9	53.9	51.7	50.0	48.5	47.3	46.2	45.6	45.0	44.6	44.3	44.1	43.9
TP-RNN w/ init vel. [21]	**84.5**	**72.0**	64.8	60.3	57.2	55.0	53.4	52.1	50.9	50.0	49.3	48.7	48.3	47.9	47.6	47.3
GRIP (Ours)	84.0	71.7	**64.8**	**60.7**	**57.8**	**55.9**	**54.2**	**52.8**	**52.4**	**51.9**	**51.6**	**51.4**	**51.3**	**51.3**	**51.2**	**51.1**

Table 4.4: Comparison to prior works using the Penn Action dataset in terms of PCK@0.05 (the higher the better). In each column, the best results are typeset in boldface and the second best are underlined.

4.3.5 Ablation Study on NGSIM Dataset

Considering that the graph representation of traffic agents are built according to the distances of objects, thus in this subsection, we do two ablation studies about our scheme in the self-driving application scenario using NGSIM datasets:

(1) We defined a threshold D_{close} in section 4.2.1. Two objects within D_{close} range are regarded as close to each other. We first explore how this threshold impacts the performance of our model. In Figure 4.3, we compare results when D_{close} is set to different values. One can see that the prediction error when $D_{close} = 0$ (when none of the surrounding objects are considered, blue bars in Figure 4.3) is higher than the results when $D_{close} > 0$ (taking nearby objects into account). Thus, considering the surrounding object indeed helps our model make a better prediction.

Also, we notice that the prediction error increases when D_{close} increases from 25 feet (orange bars) to 50 feet (green bars). This is because more objects are involved when predicting the motion of an object. In real life, a traffic agent is more likely to be only impacted by its closest objects. Thus, considering too many surrounding objects does not help to improve the prediction accuracy. Based on this observation, in this chapter, we set $D_{close} = 25$ feet as our default setting unless specified otherwise.

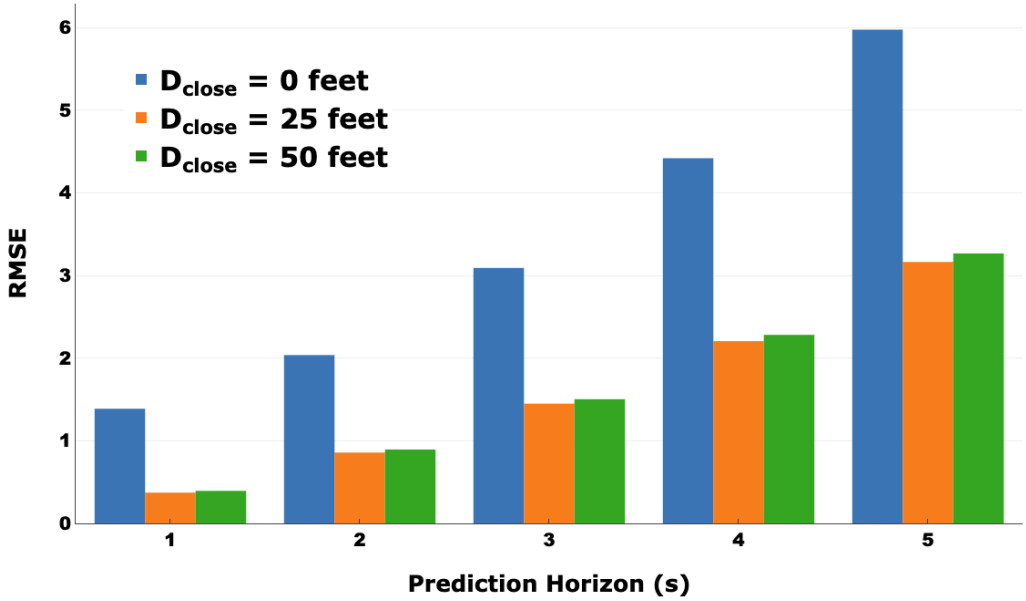


Figure 4.3: Comparison among various D_{close} values.

(2) Given an input stream consisting of observed objects' past trajectories, our model is able to predict future trajectories for all observed objects. Thus, in Figure 4.4, we report the prediction error for objects at different locations, e.g., -60 or -45 feet, within the observed area. In Figure 4.4, traffic agents are moving from location -90 to location 90 (left to right).

First, one may notice that the prediction error decreases from location -90 to -45, and then increases after -45. Such an observation is obvious on the top 3 curves ("Future 5/4/3 second"). This is impacted by the clue information from surrounding objects. Because objects are moving from left to right in Figure 4.4, so objects located at 90 can only observe objects behind them, while objects at -90 can only see objects in front of them. Thus, prediction error at -90 is lower than the error at 90 concludes that front objects are more important than behind objects for our trajectory prediction model. This is also the reason why prediction error increases after -45 (less and less front objects are observed from left to right).

In addition, considering that predicting the motion of an object in far future is difficult. Thus, in Figure 4.4, the error of a long time prediction is higher than a shorter time prediction (Curve "Future 5 second" is above curve "Future 1 second").

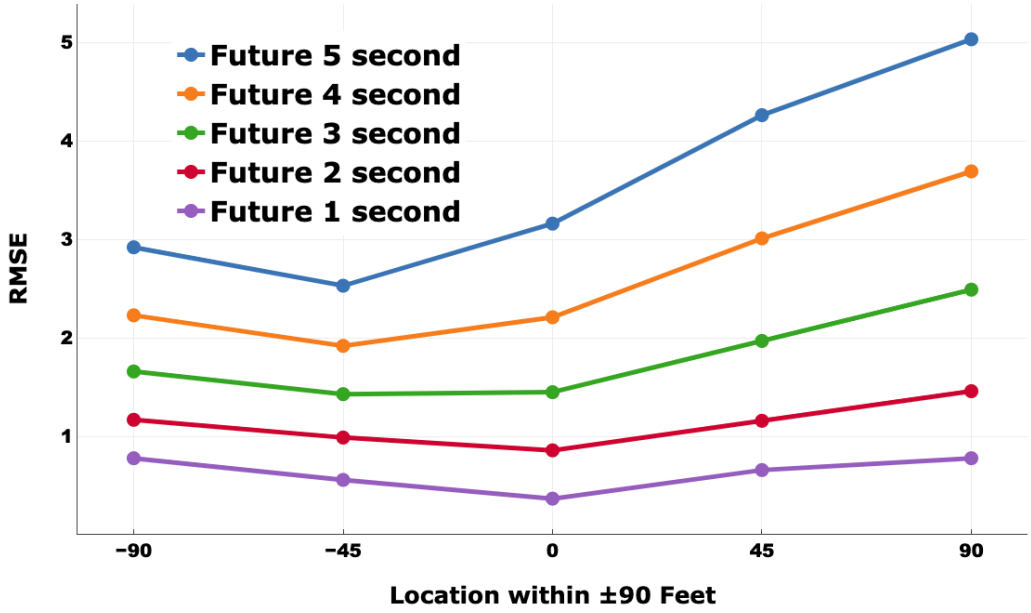


Figure 4.4: Prediction error at different locations.

4.3.6 Comparison Results on NGSIM Dataset

In this subsection, we compare our proposed scheme to the following baselines (as done in [35]) and some existing solutions using NGSIM dataset:

- Constant Velocity (CV): This is a baseline that only uses a constant velocity Kalman filter to predict trajectories in the future.
- Vanilla LSTM (V-LSTM): A baseline that feeds a tack history of the predicted object to an LSTM model to predict a distribution of its future position.
- C-VGMM + VIM: In [34], Deo et al. propose a maneuver based variational Gaussian mixture model with a Markov random field based vehicle interaction module.
- GAIL-GRU: Kuefler et al. [82] use a generative adversarial imitation learning model for vehicle trajectory prediction. However, they use ground truth data for surrounding vehicles as input during prediction phase.
- CS-LSTM (M): This is the model that an LSTM model with convolutional social pooling layers proposed by Deo et al. in [35]. A maneuver classier is included.

- CS-LSTM: A CS-LSTM model without the maneuver classifier described in [35].

Comparison results are reported in Table 4.5. Our model can predict the trajectories for all observed objects simultaneously, while other schemes listed in Table 4.5 only predict one specific object (in the middle position) each time. Thus, to make a fair comparison, we compute the RMSE for the same objects as other schemes and report the result in the second column on the right side, "GRIP (△CS-LSTM)", of Table 4.5. Compared to the existing state-of-the-art result (CS-LSTM [35]), our proposed GRIP improves the prediction performance by at least 28%. One may notice that, after 3 seconds in the future, the prediction error of GRIP is a half meter (or longer) shorter than CS-LSTM [35]. We believe that such an improvement can help an autonomous driving car avoid many traffic accidents.

Prediction Horizon (s)	CV	V-LSTM	C-VGMM + VIM [34]	GAIL-GRU [82]	CS-LSTM(M) [35]	CS-LSTM [35]	GRIP (△CS-LSTM)	GRIP (ALL)
1	0.73	0.68	0.66	0.69	0.62	0.61	**0.37** (40%↑ -0.24)	0.64
2	1.78	1.65	1.56	1.51	1.29	1.27	**0.86** (32%↑ -0.41)	1.13
3	3.13	2.91	2.75	2.55	2.13	2.09	**1.45** (31%↑ -0.64)	1.80
4	4.78	4.46	4.24	3.65	3.20	3.10	**2.21** (29%↑ -0.89)	2.62
5	6.68	6.27	5.99	4.71	4.52	4.37	**3.16** (28%↑ -1.21)	3.60

Table 4.5: Root Mean Square Error (RMSE) for trajectory prediction on NGSIM I-80 and US-101 datasets. Data are converted into the meter unit. All results except ours are extracted from [35]. The smaller the value, the better.

Besides, we also report RMSE results for all predicted objects in the last column, "GRIP (ALL)", of Table 4.5. It is worth highlighting that:

- All schemes in Table 4.5 take the same data (an object in the middle position and its surrounding objects) as their inputs. Our model predicts all observed objects simultaneously, while others only predict the one in the central location.

- As we discussed the ablation study subsection 4.3.5, objects located at the edge of the observed area, e.g., located at ± 90 feet position, do not have enough surrounding objects as input. Thus, the prediction errors of these objects are high, which results in the results in the column of "GRIP (ALL)" are higher than "GRIP".

Even so, our proposed GRIP still achieves better prediction results than all of the other existing solutions.

Then, compared the result of CS-LSTM(M) to CS-LSTM, one can see that CS-LSTM makes slightly better prediction than CS-LSTM(M). This is consistent with our argument mentioned in Chapter 1 that a wrong classification of maneuver type has an adverse effect on the trajectory prediction.

4.3.7 Computation Time

Computation efficiency is one of the important performance indicators of an algorithm for autonomous driving cars. Thus, we evaluate the computation time of our proposed GRIP and report the results in Table 4.6.

To make a fair comparison, we downloaded the code of CS-LSTM [35] [ii] and ran it on our machine to collect its computation time. Both CS-LSTM and GRIP are implemented using PyTorch.

Scheme	Predicted #	Time (s) 128 batch	Time (s) 1 batch
CS-LSTM [35]	1000	0.29	35.13
GRIP	1000	**0.05**	**6.33**

Table 4.6: Computation time

From Table 4.6, one can see that, when using 128 batch size, CS-LSTM [35] needs 0.29s to predict trajectories for 1000 objects, while our proposed GRIP only takes 0.05s (5.8x faster). In the autonomous driving application scenario, considering the limited resources, we can only set $batch\_size = 1$, so we report the results in the last column of Table 4.6. It shows that GRIP can still run 5.5 times faster than CS-LSTM [35].

4.3.8 Visualization of Prediction Results

In Figure 4.5, we visualize several prediction results in mild, moderate, and congested traffic conditions (from left to right) using the datasets NGSIM I-80 and US-101. After observing 3 seconds of history trajectories, our model predicts the trajectories over 5 seconds horizon in the future. From Figure 4.5, one can notice that:

- 1. From Figure 4.5a to Figure 4.5c, it is obvious that green-dashed lines (CS-LSTM) are longer than yellow-dashed lines (ours) and farther from the red-dashed lines (ground truth).

[ii] https://github.com/nachiket92/conv-social-pooling

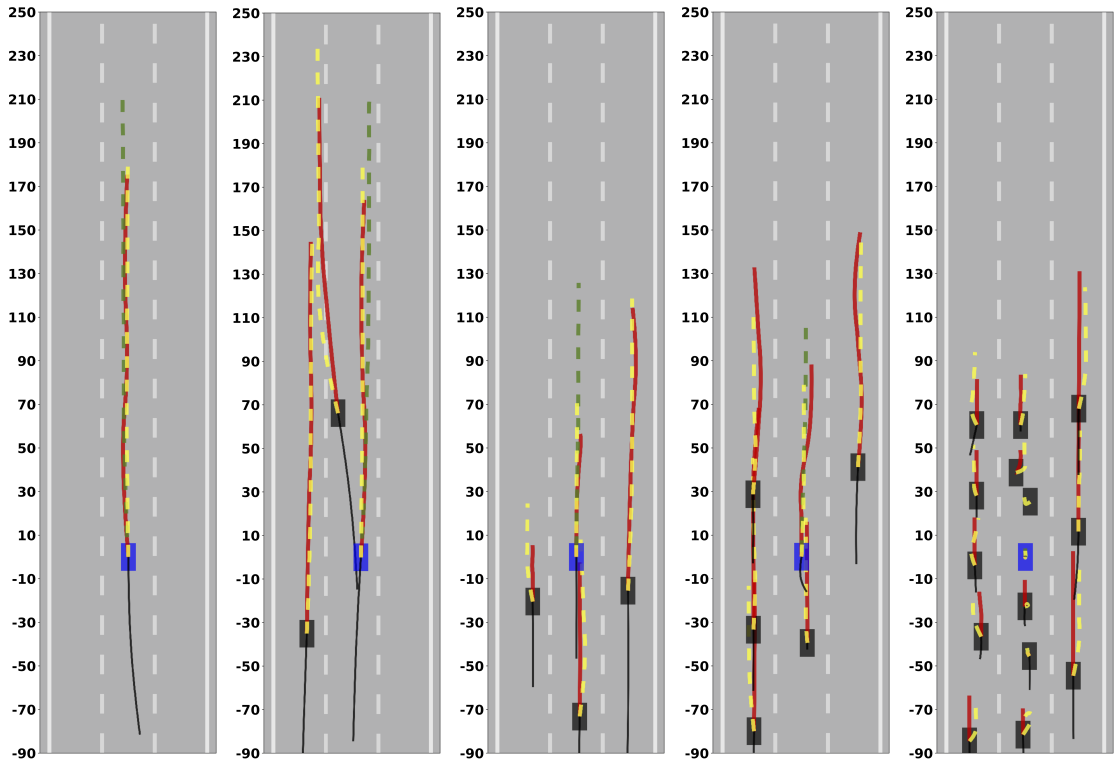


Figure 4.5: Visualized Prediction Results. Blue rectangles are the cars located in the middle which is the car that CS-LSTM [35] trys to predict. Black boxes are surrounding cars. Black-solid lines are the observed history, red-dashed lines are the ground truth in the future, yellow-dashed lines are the predicted results (5 seconds) of our GRIP, and the green-dashed lines are the predicted results (5 seconds) of CS-LSTM [35]. Region from -90 to 90 feet are observed areas.

This proves that when feeding the same history trajectories (all objects in the scene) to models, our proposed GRIP makes a better prediction for the central object than CS-LSTM.

- 2. In Figure 4.5b, our model precisely predicts the trajectory of the top car even when it is going to change lane in the next 5 seconds. In addition, the car in the left lane is affected by the top car, and our model still successfully predict the trajectory for the car in the left lane.

- 3. Our proposed GRIP can predict all objects in the scene simultaneously, while CS-LSTM can only predict the one located in the middle. Especially, in Figure 4.5e, we show a prediction result in a scene that involves 15 cars. In this scene, although some cars move slowly (vehicles in the middle lane) while others move faster (cars in the right lane), our proposed GRIP model is able to predict their future trajectories correctly and simultaneously.

Based on these observations from the visualized results, we can conclude that our proposed scheme, GRIP, indeed improves the trajectory prediction performance compared to the existing methods.

4.4 Discussion

In this chapter, we propose a novel scheme (GRIP) for the object trajectory prediction. Compared to some existing specifically designed solutions, our GRIP is a general model that can be used for human motion prediction, traffic agent trajectory prediction, and other trajectory prediction related tasks. Besides, unlike some existing solutions that only predict the future trajectory for a single traffic agent each time, GRIP is able to predict trajectories for all observed objects simultaneously. GRIP enjoys these advantages because it uses a graph to represent the interaction among all connected (or close) objects and employs an encoder-decoder LSTM model to make predictions. The experimental results on four well-known public datasets (two for human motion prediction and two for traffic trajectory prediction) show that our proposed model achieves much better prediction results than existing methods and run 5 times faster than the state-of-the-art schemes.

Currently, GRIP only utilizes the LIDAR data. In some scenarios, there are multiple sensors, e.g. RGB cameras, ultrasonic sensors. GRIP can be enhanced to utilize multiple sensor data streams to improve further its prediction accuracy and also to handle the varying environmental situations where the data stream from one particular sensor type may not be available or too sparse. In addition, researchers can explore if information extracted from LIDAR can be replaced by information extracted from other sensor types since LIDAR is expensive. For example, researchers in a recent paper [165] proposed to convert image-based depth maps to pseudo-LIDAR representations to improve 3D object detection for autonomous driving. Interested researchers can explore how GRIP performs using pseudo-LIDAR information.

Chapter 5

Data-free Automatic Acceleration of Convolutional Networks

Many human activity recognition and human motion prediction models are suffering from slow computation time, especially when we run them on mobile devices. As shown in Table 5.1, convolutional operations (CNN1 and CNN2) in both SBGAR and ReHAR take up the most of the time. Consistently, 2D convolution (Conv2D) also takes the longest time among all operations of GRIP. Thus, we argue that if we can speed up the convolutional processes, we can speed up all of our proposed schemes.

	SBGAR (10 frames)	**ReHAR (10 frames)**
Process	Computation time (ms)	Computation time (ms)
De-shake	2.42	2.42
Optical Flow image	19.77	19.77
Extract CNN1 Feature (VGG16)	**40.41**	**40.41**
Extract CNN2 Feature (VGG16)	**40.41**	**40.41**
Caption/Representation Generation	28.63	0.19
Activity Recognition	2.15	0.46
Total Time	133.79	103.66

GRIP	Conv2D	Graph Operation	Encoder LSTM	Decoder LSTM	Total Time
Computation time (ms) 1 step prediction	**4.44**	1.31	0.60	0.79	7.14

Table 5.1: Computation time of all proposed schemes.

To achieve this goal, in this chapter, we propose a deep learning model decompression solution to accelerate convolutional operations. Some related works are summarized in section 5.1. In

section 5.2, we describe the architecture of DAC and our factorization method. The experimental results are discussed in section 5.3, followed by the conclusion in Section 5.4.

5.1 Background

Much work has been done to do parameter decomposition. In this section, we will discuss some prior work that decomposes convolutional layers. To simplify the description, we assume the weight of the convolutional layer that we are going to decompose has a size of $(n \times k_w \times k_h \times c)$, where n is the number of kernels, k_w and k_h are the spatial width and height of a kernel respectively, and c is the number of channels of the input feature map.

First, Jaderberg et al. [68] propose a spatial decomposition method. The method decomposes a convolutional layer with $(n \times k_w \times k_h \times c)$ kernel size into two layers. One has horizontal filters with $(c' \times k_w \times 1 \times c)$ kernel size and the other consists of vertical filters with $(n \times 1 \times k_h \times c')$ kernel size, where c' is a new channel number for these two layers. In theory, this method indeed reduces parameters. However, running the decomposed model on a mobile device that has limited resources does not result in a significant speed up. This is due to the caching behavior of data. A feature map is horizontally (or vertically) loaded into a continuous block of memory. When we compute convolution using horizontal (vertical) filters, we access the memory sequentially. There is no impact on running time. However, if we compute the convolution using vertical (horizontal) filters, we cannot access memory sequentially any more which results in more cache misses and hence longer computation time.

Then, Zhang et al. describe a channel decomposition method in [185]. It decomposes a convolutional layer with $(n \times k_w \times k_h \times c)$ kernel size into a convolutional layer with fewer output channels and a pointwise convolutional layer. The newly generated convolutional layer has $(c' \times k_w \times k_h \times c)$ kernel size, and the pointwise convolutional layer has $(n \times 1 \times 1 \times c')$ kernel size. Notice that the first layer is also an ordinary convolutional layer, so it does not improve the situation fundamentally.

Direct tensor decomposition methods including CP decomposition [89] and Tucker decomposition [76] are also applied to accelerate networks. After these tensor decompositions, one convolution layer will be factorized into 3 or 4 small layers with a bottleneck structure, opposite with

[135] architecture. One big disadvantage of these tensor decomposition methods is that the depth of network architecture is tripled (3x) compared to the original model, thus it increases the memory access cost (MAC) and largely offset the gains from the reduction of FLOPs, as claimed in [111].

There are also many network decomposition works using low rank constraints in training process or solving layer-wise regression problem with data samples [168, 2]. But all these methods require the access of sufficient data from training/test domain. Our research focus is based on the real application scenario with limited access of data. In this chapter, we propose a novel data-free convolutional layers decomposition method and compare its performance to two most related works [185, 68].

5.2 Proposed Solution

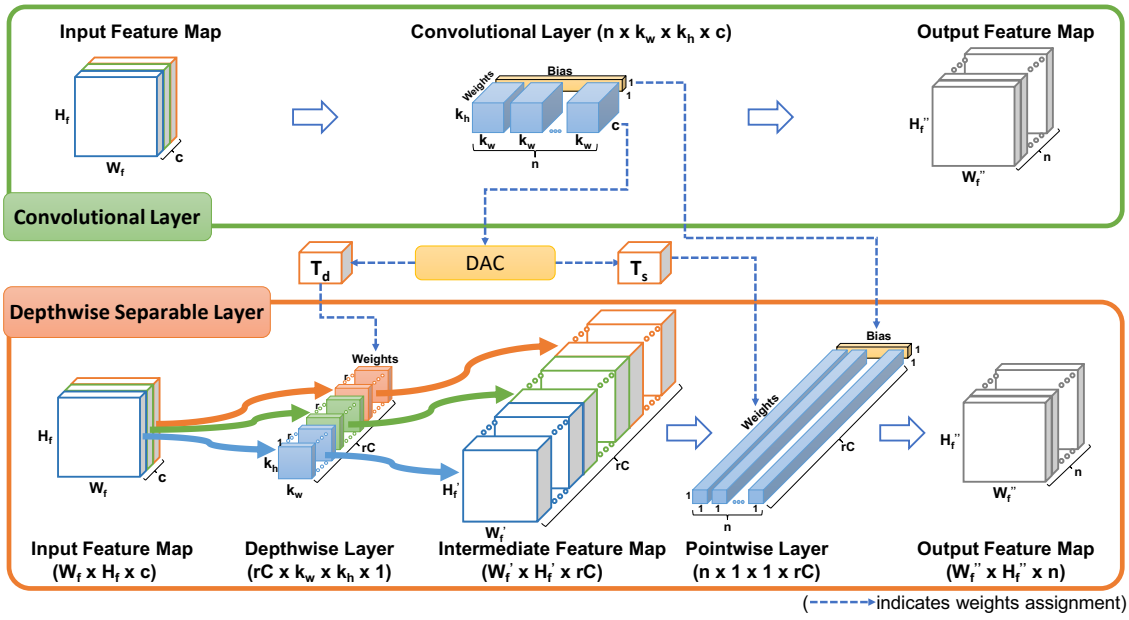


Figure 5.1: The architecture of our proposed DAC. An input feature map consists of c channels (in this figure, $c = 3$) is marked with different colors. In "Depthwise Layer", kernels are only applied on the channel with the same color. Thus, each channel is processed by r kernels.

The intuition of our proposed scheme is that the depthwise + pointwise combination runs efficiently on mobile devices has already been proven by MobileNet [62]. It will be useful if we can convert an ordinary convolutional layer into such a structure and compute their weights from the original layer directly. The feasibility of decomposing the weights of a convolutional layer has

been mathematically proved by Zhang et al. [185].

5.2.1 Convolutional Layer Factorization

In this section, we propose a novel factorization method for convolutional layers. Figure 5.1 shows the details of our scheme. An ordinary convolutional layer with the shape of $(n \times k_w \times k_h \times c)$ is decomposed into two layers. One is a depthwise layer with the shape of $(rC \times k_w \times k_h \times 1)$, and the other is a pointwise layer with the shape of $(n \times 1 \times 1 \times rC)$, where $rC = r * c$ and r is a factor used to balance the trade-off between model compression ratio and accuracy drop. There is no bias in the depthwise layer, and the bias vector in the original layer is assigned to the pointwise layer.

Even though our scheme is inspired by MobileNet, it is worth highlighting the differences between MobileNet and DAC. DAC has no non-linear layers (batch normalization layers and activation layers) between the depthwise and the pointwise layers. The absence of non-linear layers makes DAC quantization friendly and hence suitable for further hardware acceleration, which Sheng et al. [141] have already experimentally verified.

5.2.2 Weights Decomposition

Once a convolutional layer is factorized, we want to compute weights for the newly generated layers (a depthwise and a pointwise layer) from the original weights directly. We assume T is the trained weights of the original convolutional layer, and its shape is $(n \times k_w \times k_h \times c)$. We denote $Td \in D := \mathbb{R}^{rC \times k_w \times k_h \times 1}$ as the weights of the depthwise layer and $Ts \in S := \mathbb{R}^{n \times 1 \times 1 \times rC}$ as the weights of the pointwise layer. Then, the objective function of factorizing a convolutional layer is:

$$\min_{Td \in D, Ts \in S} \|T - Ts * Td\|_F^2, \tag{5.1}$$

where operator $*$ is the combination of convolution operations of the depthwise and the pointwise layer, and $\|\|_F$ is the Frobenius norm for tensor/matrix. Thus

$$
\begin{aligned}
&\min_{Td\in D, Ts\in S} \|T - Ts * Td\|_F^2 \\
&= \min_{Td\in D, Ts\in S} \sum_{i=1}^{C} \|T_i - Ts_i * Td_i\|_F^2 \\
&= \sum_{i=1}^{C} \min_{Td_i, Ts_i} \|T_i - Ts_i * Td_i\|_F^2 \\
&= \sum_{i=1}^{C} \min_{S_i, D_i} \|M_i - S_i D_i\|_F^2.
\end{aligned}
$$

Here matrices M_i, S_i and D_i are transformed from tensors T_i, Ts_i and Td_i respectively.

According to the SVD theory, the solution of minimization problem $\min_{S_i, D_i} \|M_i - S_i D_i\|_F^2$ is the singular matrices with rank r, where the top r singular values can be merged into either S_i or D_i. Also, Frobenius norm $\|\|_F$ can be defined as $\|\|_{2,2}$ induced by L_2 vector norm, so the above DAC minimization objective function can be considered as

$$
\begin{aligned}
&\min_{Td\in D, Ts\in S} \|T - Ts * Td\|_F^2 \\
&= \min_{Td\in D, Ts\in S} \sup_{\|F\|_2 \neq 0} \frac{\|(T - Ts * Td)F\|_2}{\|F\|_2},
\end{aligned}
$$

where F is the input feature maps and $\|F\|_2$ is the vector L_2 norm. In this formula, it minimizes the output feature maps with approximation error measured in Euclidean space and the constraint of the decomposition 'rank' r (the factor used to balance the trade-off between model compression ratio and accuracy drop). The process of weights decomposition is described in Algorithm 1.

5.2.3 Computation Reduction

We consider the original convolutional layer with $(n \times k_w \times k_h \times c)$ kernel size takes a $(W_f \times H_f \times c)$ feature map F as an input and produces a $(W_f \times H_f \times n)$ feature map G, where W_f and H_f are the spatial width and height of the feature maps. Here, we assume the output feature map has the same spatial size as the input for simplification. Then, the computation cost of the convolutional layer is: $W_f \times H_f \times c \times k_w \times k_h \times n$.

The computation cost depends on the number of input channels c, the number of output chan-

Algorithm 1: DAC Weights Decomposition

Input : Weights of a convolutional layer: $T \in \mathbb{R}^{n \times k_w \times k_h \times c}$;
Decomposition Rank: r.
Output: Weights of the depthwise layer: $Td \in \mathbb{R}^{rC \times k_w \times k_h \times 1}$;
Weights of the pointwise layer: $Ts \in \mathbb{R}^{n \times 1 \times 1 \times rC}$

begin
  $list\_d \in \mathbb{R}^{c \times r \times k_w \times k_h \times 1} \leftarrow \emptyset$
  $list\_s \in \mathbb{R}^{n \times 1 \times 1 \times r \times c} \leftarrow \emptyset$
  for $i \in c$ **do**
    $T_i \leftarrow T[:, :, :, i] \in \mathbb{R}^{n \times k_w \times k_h}$
    $M_i \leftarrow Reshape(Ti, (n, k_w \times k_h)) \in \mathbb{R}^{n \times k_w k_h}$
    $D_i, S_i \leftarrow Decompose(M_i, r)$
    $list\_d[i, :, :, :, :] \leftarrow D_i \in \mathbb{R}^{r \times k_w \times k_h \times 1}$
    $list\_s[:, :, :, :, i] \leftarrow S_i \in \mathbb{R}^{n \times 1 \times 1 \times r}$
  $Td \leftarrow Reshape(list\_d, (r \times c, k_w, k_h, 1))$
  $Ts \leftarrow Reshape(list\_s, (n, 1, 1, r \times c))$
function Decompose(M, r)
begin
  $U, Sigma, V \leftarrow SVD(M)$
  $Ur \leftarrow U[:, : r] \in \mathbb{R}^{n \times r}$
  $Vr \leftarrow V[: r, :] \in \mathbb{R}^{r \times k_w k_h}$
  $Sr \leftarrow Sigma[: r, : r] \in \mathbb{R}^{r \times r}$
  $D \leftarrow Reshape(Vr, (r, k_w, k_h, 1))$
  $S \leftarrow Ur\, Sr$
  $S \leftarrow Reshape(S, (n, 1, 1, r))$
  return D, S

nels n, the kernel size $k_w \times k_h$ and the input features map size $W_f \times H_f$. After decomposition, the newly generated depthwise and pointwise layer in total have the cost of $W_f \times H_f \times k_w \times k_h \times rC + W_f \times H_f \times rC \times n$, where $rC = r * c$ and the reduction in computation is

$$\frac{W_f \times H_f \times k_w \times k_h \times rC + W_f \times H_f \times rC \times n}{W_f \times H_f \times c \times k_w \times k_h \times n}$$
$$= \frac{r}{n} + \frac{r}{k_w k_h}$$

5.3 Experimental Results

To prove the universality of our proposed scheme, we apply DAC to four major application scenarios in the field of Computer Vision: (1) Image Classification, (2) Object Detection, (3) Multi-person Pose Estimation, and (4) Human Activity Recognition. We implement our scheme using Python and Keras Library [24] with Tensorflow backend [1].

5.3.1 Datasets

Five datasets are used in this chapter:

CIFAR-10 dataset: The CIFAR-10 dataset [81] consists of 50,000 training images and 10,000 test images in 10 categories. It is a small dataset, from which we can quickly get results after tuning parameters. Thus, we use it for ablation study to get some insights about DAC, e.g., the impacts of using different ranks or decomposing different layers.

ImageNet dataset: The ImageNet dataset [134] has 50,000 ILSVRC validation images in 1,000 object categories. We use this ILSVRC validation subset to evaluate the performance of DAC in the task of image classification.

Pascal VOC2007 dataset: For object detection task, Pascal VOC2007 dataset [41] is used. It consists of 4,952 testing images for object detection. The bounding box and label of each object from twenty target classes have been annotated. Each image has one or multiple objects.

Microsoft COCO dataset: The Microsoft COCO dataset [101] is used to evaluate the performance of DAC in the task of multi-person pose estimation. We use the COCO 2017 keypoints subset which consists of 5,000 validation images and 40K testing images.

UCF Sports Action Dataset: The UCF Sports dataset[i] [132] is used to evaluate the performance of DAC in the task of human activity recognition. This dataset consists of a set of actions collected from a wide range of stock footage websites including BBC Motion gallery and GettyImages. It consists of a total of 150 videos. Each video has one of these 10 action categories: diving, golf swing, kicking, lifting, riding horse, running, skateboarding, swinging-bench, swinging-side, and walking.

5.3.2 Ablation Study

Here, we use a pre-trained CIFAR-VGG model[ii], a simple Convolutional Neural Network, on the CIFAR-10 dataset as our original model. Figure 5.2 shows the architecture of the CIFAR-VGG. In total, the CIFAR-VGG model has 13 convolutional layers. The original model (trained on CIFAR-10 training subset) achieves 93.6% on CIFAR-10 testing subset.

[i] http://crcv.ucf.edu/data/UCF_Sports_Action.php

[ii] https://github.com/geifmany/cifar-vgg

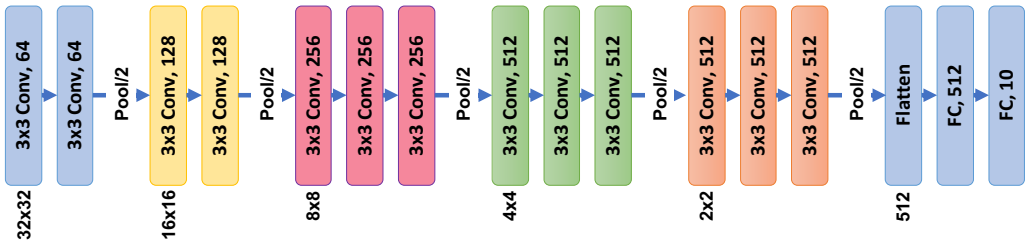


Figure 5.2: The architecture of the CIFAR-VGG.

First, we decompose a single convolutional layer to explore the impact of decomposing different layers. Table 5.2 shows the details of testing accuracy when applying varying ranks (rank 1 to rank 5) decomposition on different layers of CIFAR-VGG model. Each time, we only modify one layer. All results are collected using decomposed weights directly (no access to data or any training process).

	Accuracy (%)				
Original Model	93.6				
Decomposed Layer	**Rank1**	**Rank2**	**Rank3**	**Rank4**	**Rank5**
conv2d_1	18.6	76.4	86.7	91.9	92.8
conv2d_2	39.1	86.5	91.6	92.7	93.2
conv2d_3	54.0	87.8	92.6	93.2	93.4
conv2d_4	31.4	83.7	91.8	92.8	93.2
conv2d_5	80.1	90.2	92.6	93.1	93.8
conv2d_6	84.3	90.9	92.8	93.3	93.4
conv2d_7	66.0	89.5	92.6	93.0	93.3
conv2d_8	83.2	91.1	92.6	93.0	93.2
conv2d_9	91.2	93.1	93.3	93.4	93.5
conv2d_10	91.7	93.2	93.3	93.3	93.4
conv2d_11	93.1	93.4	93.3	93.4	93.4
conv2d_12	93.3	93.4	93.4	93.4	93.5
conv2d_13	92.9	93.4	93.4	93.4	93.5

Table 5.2: Testing Accuracy on CIFAR-10 dataset when decomposing different layers of CIFAR-VGG model using variant ranks.

From Table 5.2, we gain two insights: (a) Decomposing first few layers of a model causes large drops in accuracy (75% drop when rank 1 decomposition is applied on layer conv2d_1), while decomposing last few layers has a smaller impact on the accuracy (less than 1% drop when rank 1 decomposition is applied on layer conv2d_13). (b) Decomposing a layer using a larger rank helps to maintain the accuracy. This can be observed by comparing different columns in the same

row. These two insights are consistent with our intuition. (a) Decomposing a layer generates tiny errors. If such errors occur at the beginning of a model, the errors will accumulate to bigger errors at the final prediction. (b) Compared to smaller ranks, larger ranks generate more parameters in the depthwise layers. Thus, the newly generated layers have more possibility of replicating the performance of the original layer.

Next, we explore the performance of DAC when multiple convolutional layers are decomposed. We decompose the model with two opposite directions: (1) from the last layer to the first one, and (2) from the first layer to the last one. To simply the experiment, we use the same rank to decompose all chosen layers. The experimental results are reported in Figure 5.3. First, one can quickly notice that most decomposition cases (solid points) achieve high accuracies (higher than 91.6% or 2% drop). Second, after saving 42% FLOPs, DAC still achieves 92.7% accuracy (drops less than 1%). Both of these prove that our proposed DAC has the capability of maintaining accuracy when the number of FLOPs is substantially reduced.

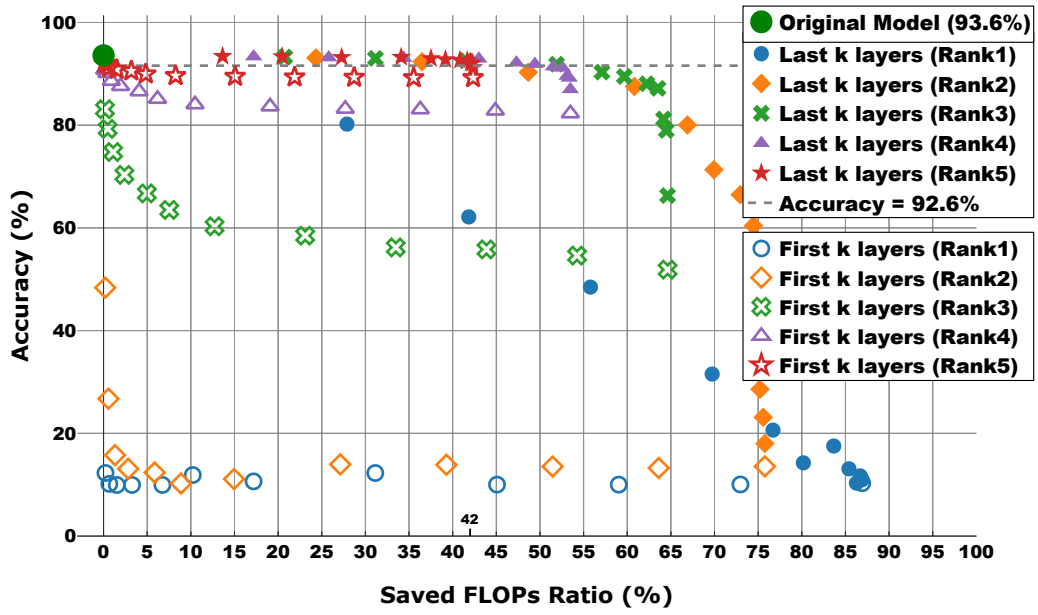


Figure 5.3: Classification accuracy on the CIFAR-10 dataset. Each curve has 12 points that correspond to different numbers of decomposed layers (2 to 13 layers from left to right). Solid spots indicate the cases that last few layers are decomposed (layer "conv2d_13" included). Open spots are the cases that first few layers are decomposed ("conv2d_1" layer included).

Besides, in Figure 5.3, red-star points (Rank 5) achieve high accuracies. If we compare the solid (open) red-star marks to other solid (open) marks, we can notice that the above insights also

hold in the case of decomposing multiple convolutional layers. Ten (eight) out of twelve Rank 5 decomposition cases (solid red-star spots) drop accuracy by less than 2% (1%). The worst solid red-star case that achieves 91.2% (accuracy drops 2.4%) is caused by the decomposition of the first layers of the model (first insight discussed above). It is worth highlighting that these decomposed models that maintain high accuracies are generated by DAC without accessing data or training process.

5.3.3 Image Classification

For the task of image classification, we use the VGG16 model proposed by Simonyan et al. in [146]. It includes 12 (3x3) convolutional layers. We downloaded a model[iii] pre-trained on ImageNet dataset. All convolutional layers but the first one are decomposed considering the first insight we got in our ablation study.

Here we compare our approach with two schemes, namely, the Filter Reconstruction Optimization proposed by Jaderberg et al. [68] (Spatial Decomp. in Table 5.3) and the Channel Decomposition method proposed by Zhang et al. [185] (Channel Decomp. in Table 5.3). Spatial Decomposition is the one that does not need data and training like DAC as we discussed in Section 5.1. Although the Channel Decomposition requires some data, we can still use the method as a filter reconstruction without accessing any data and training process. We implemented these two algorithms ourselves. For fair comparison, we choose appropriate parameters for Channel Decomposition and Spatial Decomposition, so that all schemes save roughly same FLOPs. Given a rank r of DAC, the number of filters c'_c in the first newly generated layer in Channel Decomposition can be computed using:

$$c'_c = r * \frac{c(n + k_h k_w)}{c k_h k_w + n} \tag{5.2}$$

and for Spatial Decomposition, the number of filters c'_s in the first newly generated layer is

$$c'_s = r * \frac{c(n + k_h k_w)}{c k_w + n k_h} \tag{5.3}$$

[iii] https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels.h5

where n is the number of kernels in original convolutional layer, k_w and k_h are the spatial width and height of a kernel respectively, and c is the number of channels of the input feature map.

	Top-5(Top-1) Accuracy (%)		
VGG16 [146] (Baseline)	88.9(69.2)		
Method	**Saved 40%**	**Saved 50%**	**Saved 60%**
Channel Decomp. [185]	86.5(65.6)	74.4(48.7)	43.3(20.8)
Spatial Decomp. [68]	88.6(68.5)	86.3(65.0)	78.0(52.5)
DAC (Ours)	**88.6(68.5)**	**87.5(66.8)**	**84.7(62.5)**

Table 5.3: Top-5(Top-1) Validation Accuracy on ImageNet dataset

Table 5.3 shows the accuracy of the model (after saving 40%, 50%, and 60% FLOPs respectively) on ImageNet validation set. First, DAC maintains high accuracy on both Top-1 and Top-5 accuracy even when a significant amount of FLOPs are reduced. Second, compared to the Channel Decomposition and Spatial Decomposition, DAC performs better. Especially when we saved 60% FLOPs, DAC achieves 41.4% higher accuracy than Channel Decomposition and 6.7% higher accuracy than Spatial Decomposition.

5.3.4 Multi-person Pose Estimation

For the task of multi-person pose estimation, we use the scheme proposed by Cao et al. [12]. Figure 5.4 is the architecture extracted from their paper. After generating the feature map F by a convolutional network (initialized by the first 10 layers of VGG-19 [146] and fine-tuned), the model is split into two branches: the top branch predicts the confidence maps, and the bottom branch predicts the affinity fields.

We download an implementation of Cao's model [iv] that was pre-trained on Microsoft COCO dataset as our original model. It achieves 57.9% average precision (AP) on the validation subset of 2017 COCO keypoints challenge. This model consists of six stages, which means $t \in \{2, 3, 4, 5, 6\}$ in Figure 5.4. Thus, the first stage (Stage 1) has 6 convolutional layers (3x3 kernel size), and each of the following stage (Stage 2 to Stage 6) includes 10 convolutional layers (7x7 kernel size). Based on the above two insights, we decompose the model from the bottom to the top with variant ranks (from Rank20 to Rank3). Because the full rank of a (3x3) convolutional kernel (in Stage 1) is 9,

[iv] https://github.com/anatolix/keras_Realtime_Multi-Person_Pose_Estimation

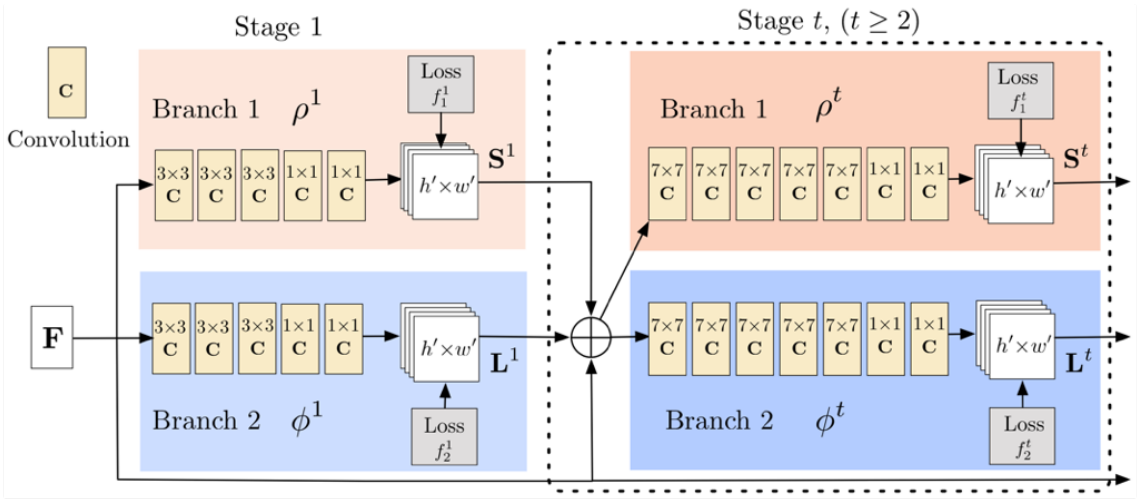


Figure 5.4: The model architecture figure extracted from [12].

so we set the maximum rank used to decompose these (3x3) convolutional layers equals to 5 for a large compression ratio.

Figure 5.5 shows the experimental results. First, it is obvious that in the task of person pose estimation, the DAC also maintains high accuracy without any retraining when large amounts of FLOPs are saved. Our proposed DAC saves up to 46% FLOPs when 2% AP drop is allowed. Second, for each curve, the AP decreases with decreasing decomposition rank. This observation is consistent with the above second insight. Then, we notice that "Decompose last 6 stages" achieves similar results (similar saved ratios and APs) as "Decompose last 5 stages" does. This can be explained as follows: the "Decompose last 6 stages" includes Stage 1 in which all decomposed convolutional layers (6 layers) have (3x3) kernel size. Comparing to a convolutional layer with (7x7) kernel size, these layers have much fewer parameters, so decomposing them does not contribute much.

Table 5.4 shows the accuracy of the model (after saving 40%, 50%, and 60% FLOPs respectively) on COCO 2017 keypoint challenge. The parameters of Channel Decomposition and Spatial Decomposition are computed using Equation 5.2 and 5.3 correspondingly. Compared to Channel and Spatial Decomposition, DAC achieves higher accuracy even when a significant amount of FLOPs is reduced. After saving 60% FLOPs, Channel Decomposition cannot correctly detect any person's pose, while DAC can still achieve 7.1% higher accuracy than Spatial Decomposition.

Figure 5.6 shows the visualized multi-person pose estimation results on COCO dataset. It

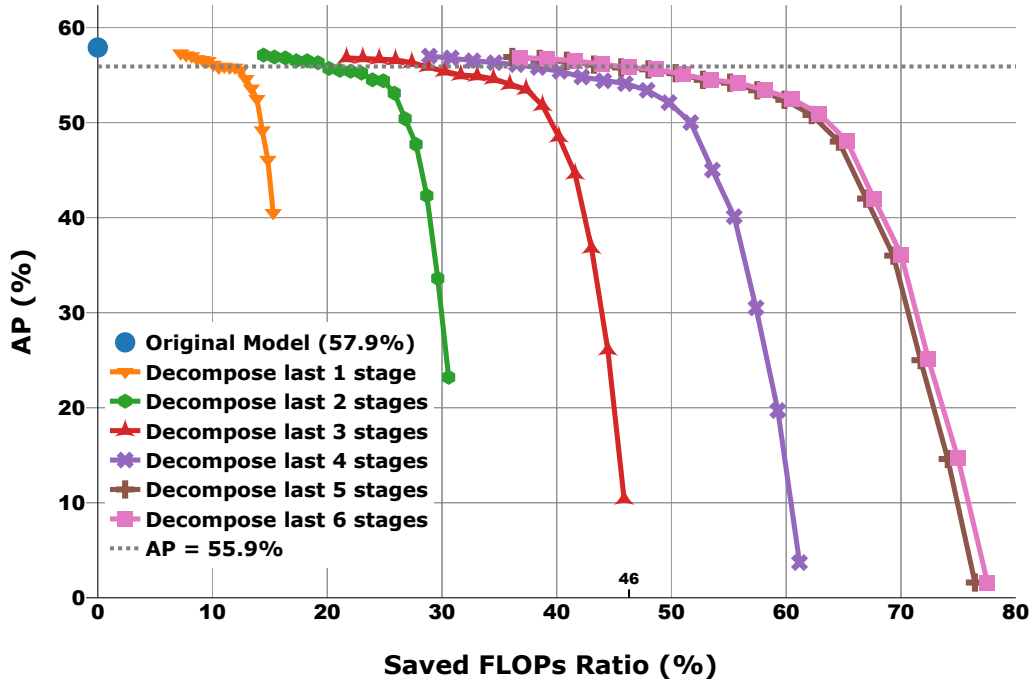


Figure 5.5: Results on the Microsoft COCO dataset. Each curve has 18 points that correspond to different ranks (Rank20 to Rank3 from left to right).

	Mean Average Precision (%)		
Openpose [12] (Original)	57.9		
Method	**Saved 40%**	**Saved 50%**	**Saved 60%**
Channel Decomp. [185]	25.9	5.0	0
Spatial Decomp. [68]	55.9	54.4	45.4
DAC (Ours)	**56.7**	**55.6**	**52.5**

Table 5.4: Results on the COCO 2017 keypoint challenge

shows that after being decomposed using DAC, the model still works pretty well. There are only small changes observed. For example, the decomposed model misses a leg of a person in the first example (the second person on the right side) and the third sample (the second person on the left side).

5.3.5 Object Detection

Next, we evaluate the performance of DAC in the task of object detection using the Single Shot MultiBox Detector (SSD) model proposed by Liu et al. [104]. Figure 5.7 shows the framework of the SSD.

Figure 5.6: Visualized results on COCO dataset. The first row shows the results generated using the original weights, while the second row shows the results created using the model that saves 50% FLOPs.

We use a model [v] pre-trained on Pascal VOC2007 and VOC2012 trainval subset. The model uses VGG-16 [146] as its base net that has (300x300) input size. The authors added ten extra convolutional layers to the VGG-16 model to provide extra information. In total, 18 (3x3) convolutional layers and 5 (1x1) convolutional layers are used to generate multi-scale feature maps for detection, and 12 (3x3) convolutional layers are used to produce a fixed set of detection predictions. This model achieves 76.5% on VOC2007 testing set.

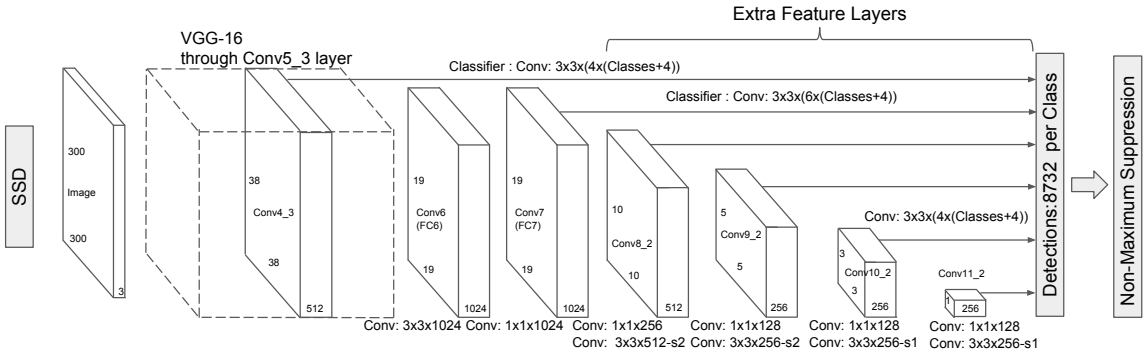


Figure 5.7: SSD architecture extracted from [104]

There is no benefit in decomposing a convolutional layer with (1x1) kernel size, so we only decompose those layers with (3x3) kernel size. Furthermore, considering that decomposing first layers causes large drops of accuracy, we do not decompose the first convolutional layer of the model.

[v] https://github.com/pierluigiferrari/ssd_keras

To simplify the description, we denote 18 layers (the first layer, conv1_1, is not decomposed) that generate multi-scale feature maps by "Feature Convolutional Layers (FL)" and 12 layers that produce detection predictions by "Detector Convolutional Layers (DL)".

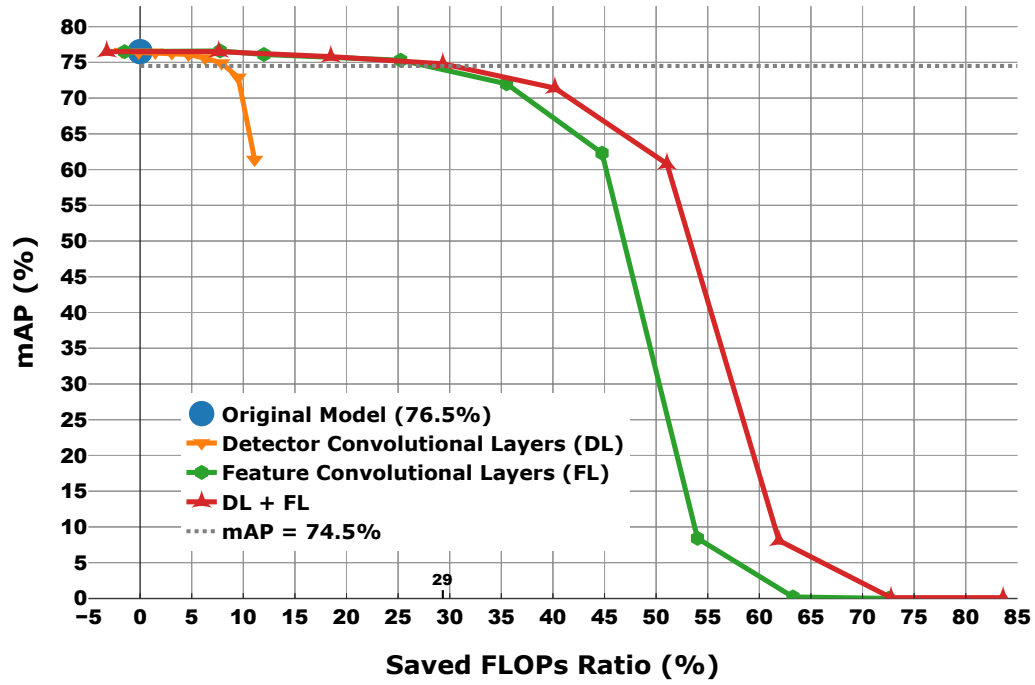


Figure 5.8: Object detection results on PASCAL VOC2007 testing set. Nine spots on each curve indicate Rank9 toward Rank1 correspondingly from left to right.

We demonstrate the experimental results in Figure 5.8. First, one can see that if 2% mAP drop is acceptable, DAC saves up to 29% FLOPs. Second, decreasing the decomposition rank results in a drop of mAP, which is also observed in the previous experiment. Third, compared to "DL", "FL" achieves a bigger FLOPs saved ratio. This is because that there are fewer layers in "DL" and each layer in "DL" has fewer channels than layers in "FL". In addition, for this model, the maximum decomposition rank is 9 so when the decomposition rank is set to 9, the number of parameters increases after decomposition. This is because that all layers we decompose in this model have (3x3) kernel size whose full rank is 9. The newly generated depthwise layer with Rank9 has the same number of parameters as the decomposed layer, while an extra pointwise layer that has $rC \times N \times 1 \times 1$ parameters is added.

Table 5.5 shows the comparison of the detection accuracy on PASCAL VOC2007 Dataset. One can see that DAC achieves higher accuracy than other schemes. In Table 5.6, we list the

	Mean Average Precision (%)		
SSD [104] (Original)	76.5		
Method	**Saved 30%**	**Saved 40%**	**Saved 50%**
Channel Decomp. [185]	62.2	60.0	52.4
Spatial Decomp. [68]	63.1	62.2	60.6
DAC (Ours)	**74.8**	**71.4**	**60.8**

Table 5.5: Object detection results on PASCAL VOC2007 Dataset.

details of the detection results on PASCAL VOC2007 testing set. Comparing the results of DAC to the original model, one can see that decomposing the model using DAC does not impact the performance of the model too much, for all categories. The change of the accuracy happens on each category within a small range.

Model	mAP(%)	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
SSD[104](Original)	76.5	78.6	83.9	75.3	67.8	48.5	86.7	84.7	87.7	58.1	79.3	75.0	85.9	87.5	82.6	77.5	51.2	77.1	79.5	87.2	76.5
(DL) Channel[185]	76.1	77.6	83.8	75.8	66.6	45.7	86.4	84.5	87.6	58.1	78.6	74.5	86.1	87.4	82.5	77.0	50.3	76.8	79.7	87.8	75.1
(DL) Spatial[68]	76.1	77.9	83.2	75.3	67.2	46.0	86.3	84.4	86.9	58.2	78.9	74.5	85.5	87.3	82.7	76.9	51.0	76.5	79.4	87.7	75.7
(DL) DAC(Ours)	**76.3**	78.4	82.9	74.5	68.3	47.8	86.7	84.4	88.4	58.0	79.4	74.9	85.6	86.5	83.1	77.3	50.7	77.3	79.0	87.5	76.1
(FL) Channel[185]	62.2	70.4	69.7	63.8	52.9	38.3	75.1	79.8	72.8	42.2	73.2	38.0	65.7	76.3	69.6	64.0	38.8	66.6	53.4	75.6	57.3
(FL) Spatial[68]	63.2	73.7	69.7	64.6	52.0	39.0	75.6	79.9	77.6	42.6	73.2	39.5	70.7	76.3	71.3	65.5	37.9	67.0	53.0	77.9	56.1
(FL) DAC(Ours)	**75.3**	78.2	83.0	73.0	67.1	44.3	86.3	83.3	87.7	56.6	78.5	75.2	84.2	85.9	82.8	75.8	48.8	75.3	78.6	86.4	75.6
(DL+FL) Channel[185]	62.2	70.6	69.4	64.1	51.1	36.1	75.7	79.8	72.8	43.0	72.9	39.9	66.4	74.5	70.2	63.7	38.5	65.9	55.1	75.4	58.7
(DL+FL) Spatial[68]	63.1	73.8	70.3	64.1	50.8	37.8	75.3	79.8	76.9	43.0	74.3	39.8	69.7	75.8	70.9	64.5	38.6	68.9	54.1	77.9	56.2
(DL+FL) DAC(Ours)	**74.8**	76.4	81.1	73.1	66.0	44.6	85.9	83.1	88.1	56.5	76.8	74.0	84.3	86.1	83.1	75.5	47.7	74.1	77.5	86.1	75.7

Table 5.6: Detection results on PASCAL VOC2007 testing set. All results expected original are collected using the SSD300 model decomposed with Rank6. "DL" indicates only Detector Convolutional Layers are decomposed and "FL" indicates only Feature Convolutional Layers are decomposed.

Figure 5.9 shows the visualized object detection results on PASCAL VOC2007 testing set. From the first two samples, one can see that after being decomposed, the model can still correctly detect objects. The locations and sizes of the detected bounding boxes have small changes. The third sample is an example that the original model does not detect an object (the bottle) that is successfully detected by our decomposed model. The fourth sample shows an extra false positive example (an unexpected potted-plant is detected), the fifth sample is a missing example (miss the car on the right), and the last sample is an example that the detected label changed (from bird to dog).

5.3.6 Human Activity Recognition

Then, we evaluate the performance of DAC in the task of human activity using our ReHAR model proposed in Chapter 3. Figure 3.1 shows its framework. We use our proposed ReHAR model that

Figure 5.9: Visualized results on PASCAL VOC2007 dataset. The first row shows the results generated using the original weights, while the second row shows the results created using a model that saves 40% FLOPs. Red dashed rectangles are ground truths. The first two samples are examples that the model works well after being decomposed, the third sample shows an example that DAC helps improve the performance, while the following three samples are different kinds of errors caused by decomposition.

achieves 90.4% 5-fold cross-validation accuracy on the UCF Sports Action Dataset. The model consists of two CNN networks (CNN1 and CNN2) for feature extraction. We decompose both CNN networks (VGG16) using DAC and report the results in Table 5.7.

	5-fold cross-validation Accuracy (%)				
ReHAR (Original)	90.4				
Method	**Saved 30%**	**Saved 40%**	**Saved 50%**	**Saved 60%**	**Saved 70%**
Channel Decomp. [185]	88.4	88.3	83.6	64.2	50.7
Spatial Decomp. [68]	90.2	88.2	88.0	76.1	58.5
DAC (Ours)	**90.3**	**90.3**	**89.8**	**85.4**	**75.5**

Table 5.7: Human activity recognition results on the Sports Action Dataset.

The experimental results show that DAC still works in the task of Human Activity Recognition. More specifically, it works for CNN+LSTM structure. Compared to two existing data-free decomposition schemes (Channel Decomposition and Spatial Decomposition), DAC maintains higher accuracy. Even after saving 60% FLOPs using DAC, the accuracy of the model only drops 5%. However, the other two schemes cause more than 10% accuracy drops. It is worth highlighting that all of these three schemes perform well on ReHAR if its saved FLOPs are not larger than 50% (Saved 30%, 40%, and 50% in Table 5.7). Usually, a decomposed CNN network generates poorer features, which should result in large accuracy drops. However, in ReHAR, we feed such poorer features to two LSTM models (LSTM1 and LSTM2), which improves the robustness of ReHAR

towards negative impacts of decomposition. Even though the extracted features are poor, the final accuracy results are still good, which proves the robustness of ReHAR.

In addition, we also report the computation time of ReHAR after being decomposed using DAC in Table 5.8. The computation time of a deep learning model depends on many factors, e.g., hardware (memory size of a GPU), model structures (with/without a loop/skip connection), hyperparameter (batch size), and so on. In this experiment, we tried our best to fix other factors and let the computation time reported in Table 5.8 be only impacted by the FLOPs of a model. From this experimental result, one can see that DAC speeds up CNN networks. More saved FLOPs results in more acceleration. More saved FLOPs results in more acceleration, but accuracy may drop, so a good tradeoff point should be chosen to meet the performance requirements of the application scenario where DAC is applied.

Process	ReHAR (Ori.)	Saved 30%	Saved 40%	Saved 50%	Saved 60%	Saved 70%
De-shake	2.42	2.42	2.42	2.42	2.42	2.42
Optical Flow image	19.77	19.77	19.77	19.77	19.77	19.77
CNN1 (VGG16)	**40.41**	**28.76**	**27.74**	**25.41**	**23.94**	**17.70**
CNN2 (VGG16)	**40.41**	**28.76**	**27.74**	**25.41**	**23.94**	**17.70**
Representation Generation (LSTM1)	0.19	0.19	0.19	0.19	0.19	0.19
Activity Recognition (LSTM2)	0.46	0.46	0.46	0.46	0.46	0.46
Total Time (ms)	103.66	80.36	78.32	73.66	70.72	58.24

Table 5.8: Computation time (ms) of ReHAR. All results are collected using 10 frames as input and average over 10 runs.

5.4 Discussion

In this chapter, we propose a novel decomposition method, namely DAC. Given a pre-trained model, DAC is able to factorize an ordinary convolutional layer into two layers with much fewer parameters and computes their weights by decomposing the original weights directly. Thus, no training (or fine-tuning) or any data is needed. The experimental results on four computer vision tasks show that DAC reduces a large fraction, e.g., 30% to 70%, of FLOPs while maintaining high accuracy of a pre-trained model.

Similar to DAC, Thakker et al. [154] propose a Hybrid Matric Decomposition to compress RNN networks. Pan et al. [120] utilize the low-rank tensor ring decomposition to accelerate an LSTM model. The proposed DAC can also be used for RNN acceleration with a small modification.

Considering an LSTM model takes an input with a size of (T, N, D), where T is the number of time steps, N is the batch size, and D is the dimension of the feature. At each step of an LSTM, the weights in the LSTM cell have a size of (D, O) (O is the output dimension). When we compress an LSTM model, we decompose weights (size is (D, O)). The size of a weight is the same size as the input of the "Decompose" function in Algorithm 1. Thus, to decompose an LSTM model, DAC only needs to take one weight in the LSTM cell as input and decompose the weight into two separate parts using the "Decompose" function. The channel loop (line 4) and reshape operations (line 6, 10, and 11) in Algorithm 1 are not needed anymore. Interested researchers can explore how this enhanced DAC performs compared to [154, 120]. In addition, it is also interesting to explore the possibility of applying DAC to other deep learning models, e.g., Reinforcement-Learning [39, 40].

Chapter 6

Conclusions and Future Work

In this thesis, we focused on the topic of human activity recognition and prediction from videos. Our specific goals are to explore more efficient algorithms with high accuracy that can be deployed on resource-constrained devices. Our contribution to the community is four novel proposed deep learning solutions. Specifically, two efficient models [94, 95] are discussed for human activity recognition, one general motion prediction scheme [96] is proposed for human motion prediction as well as motion planning in self-driving cars, and one novel model decomposition scheme [97] is proposed to accelerate deep learning-based models. In the rest of this chapter, we will summarize each of these proposed schemes and discuss potential future work that other researchers can work on.

6.1 Summary

Automatically recognizing human activities from videos is a challenging and fundamental task in the field of computer vision. Designing an efficient and robust recognition system is non-trivial. Creating a model that performs well for both single-person activity recognition and group activity recognition under the experimental environment is challenging, not to mention designing a general model that is robust to variation of videos. Different from most of the current state-of-the-art solutions which mostly are designed for recognizing single-person events, in Chapter 2, we propose a novel scheme, namely SBGAR, that can be used to recognize both single-person activities and group activities. We initially suggest using semantic representation for the human activity

recognition task by generating a semantic caption for every single frame of the input video, and then recognize activities according to these captions. The benefits of our SBGAR is two-fold. First, SBGAR is an efficient scheme as a result of not using time-consuming models, e.g., object detection or tracking models. SBGAR runs roughly 4x faster than the state-of-the-art solution [64]. Second, SBGAR is robust to the intermediate errors (wrong semantic captions) caused by the variations of videos. Because our SBGAR generates an independent representation for each frame, thus the model can correct an intermediate error by gathering continuous representations. Experimental results on two well-known datasets show that SBGAR performs better (improves 3% and 15% accuracy separately) than the state-of-the-art method. Both characteristics make SBGAR outperforms the existing solutions.

In Chapter 3, we propose a faster and more scalable scheme based on self-examination of our SBGAR. After carefully analyzing SBGAR, we notice some of its limitations. (1). We cannot make sure that our SBGAR always generates a perfect caption for a video frame. It negatively impacts the accuracy of our model to some extent. (2). The caption generation model was not trained for the final recognition task. It leaves room for us to improve. (3). SBGAR requires a large number of labeled semantic captions, which results in SBGAR not being scalable for all application scenarios. To enhance SBGAR, we propose a new model, namely ReHAR, by replacing the semantic representation with an intermediate probability distribution representation, so the whole model is end-to-end trainable for the final activity recognition. The comparison between ReHAR and SBGAR proves that the end-to-end trainable structure indeed helps ReHAR extract more distinguishing features and achieves higher accuracy. ReHAR runs 30% faster than SBGAR. In addition, the visualization results prove that ReHAR pays more attention to the moving part of the input frames, which is consistent with our expectation.

Considering that a human activity recognition scheme can only perform well after an activity has finished, we propose a graph-based motion prediction scheme, called GRIP, in Chapter 4 to break this limitation. GRIP predicts the motion intent of an observed object considering the motion impact of its nearby objects. Such motion impact is represented using a graph of interactions of close objects. Unlike most existing work [91, 130, 46, 19, 35], GRIP can predict multiple trajectories simultaneously, which results in GRIP running 5x faster than the state-of-the-art scheme [35].

Because of using a graph to consider the impacts of nearby objects, GRIP improves the prediction accuracy of the state-of-the-art solution by 30%.

Nowadays, more and more deep learning models have been deployed on resource-constrained devices, e.g., smartphones or self-driving cars. The efficiency has become one of the most important factors of a deep learning model. To accelerate deep learning-based models, including the models we proposed in this thesis, we propose a data-free deep learning model acceleration scheme, namely DAC, in Chapter 5. Given a pre-trained deep learning model, DAC automatically divides a standard convolutional layer into one depthwise layer and one pointwise layer. The weights of both newly generated layers are calculated directly from the original layer. Thus, neither data nor training process is needed by DAC. The experimental results on four computer vision application scenarios show that DAC maintains high accuracies even when a vast amount of FLOPs is trimmed. Specifically, compared to Channel Decomposition, even when 60% FLOPs are decomposed, DAC maintains 41% higher accuracy for ImageNet classification task, 52% higher mAP for COCO2017 Pose Estimation challenge, and 20% higher accuracy in human activity recognition task. Similarly, compared to Spatial Decomposition, DAC achieves 6%, 7%, and 9% higher accuracy in the task of image classification, pose estimation, and human activity recognition.

Through this thesis, readers learned characteristics and challenges in the design of human activity recognition and prediction. We also discussed some work in the field of deep learning-based model acceleration. By reading this thesis, readers should gain some sense of critical factors while designing an efficient algorithm that can be deployed on resource-constrained devices. Apart from it, we hope some methodologies proposed in this thesis can inspire readers. In the next section, we will discuss some future work.

6.2 Future work

For the task of human activity recognition, there is still room for improvement. For example, one can use other layers, e.g., 3D convolutional [157] or 2+1D convolutional [158] layers, to replace 2D convolutional layers in our model to extract more meaningful spatiotemporal features from videos. As we discussed in Chapter 1, some research claims 3D or 2+1D convolutional networks perform better than a 2D CNN model, thus using these specific networks as our base net may achieve higher

accuracy. For seeking a faster speed, one can try to use a lighter network, e.g., MobileNet [62], as a base net in our proposed solutions. Compared to other CNN models, MobileNet runs much faster while maintaining accuracy. Thus, using MobileNet as our base net should speed up our model without losing too much accuracy. Moreover, given sufficient GPU resources, one can explore the performance of our proposed schemes, SBGAR and ReHAR, on larger datasets, e.g. UCF101 [147] and THUMOS[72].

Recently, instead of detecting persons in a video frame, researchers [4] propose to train a model that can generate initial activity heat maps by using the bounding boxes of persons during the training phase. However, their scheme uses several refinement stages to create these heat maps, which causes their solution to run much slower than our schemes (SBGAR and ReHAR). In [186], Zhao et al. propose to convert a two-stream structure into a two-in-one architecture for human action detection. Specifically, they combine spatial features (extracted from RGB frames) and temporal features (extracted from optical flow images) at several intermediate layers of the model. Although both SBGAR and ReHAR can extract features from individual persons (SBGAR predicts activities for every person, and the visualized results of ReHAR prove this statement), other researchers can also enhance them by using bounding boxes of persons while training the model. Adding bounding boxes as grounding truths will force the model to pay more attention to some specific regions of frames, instead of letting the model learn everything by itself. Thus, it is possible to use a lighter model to achieve similar or even better results. In addition, other researchers can explore the capibility of SBGAR and ReHAR in detecting actions (locating actors) from videos, considering that the proposed schemes have already focused on actors.

Besides activity recognition, the two proposed human activity recognition schemes, SBGAR and ReHAR, can also be used for activity detection, abnormal detection, video description, video retrieval, and other related tasks, with tiny modification. It is because that our proposed SGBAR (ReHAR) generates a semantic caption (an intermediate probability distribution representation) for every single video frame, one can easily locate the starting and ending points of activities from an untrimmed video by analyzing the generated captions.

In this thesis, GRIP is built using either fixed pre-designed graph (e.g., human body skeleton) or a manually designed rule (e.g., the distance of traffic agents) to generate a graph to represent

the connection of objects. However, this may become a limitation of GRIP in other application scenarios because the pre-designed graphs or rules may not be suitable for all conditions. For example, two hands are not connected according to human body skeleton; however, they are highly related for activity recognition and motion prediction. Thus, one can explore a solution that allows GRIP to learn the graph by itself and adapts based on its inputs. In our GRIP solution, we only utilize the locations of objects, but other researchers explore multimodal approach where both RGB and LIDAR data are used for the prediction task. Thus, one can also extend the proposed model by adding visual data collected using RGB cameras, etc. to further improve the prediction performance. In order to make sure any new trajectory prediction scheme can work in different scenarios, such scheme should be evaluated using more datasets, e.g., the newly released Appollo dataset [112], which captures data not only on a highway but also from urban areas.

In addition, if we combine SBGAR and GRIP, our model can be used for generating stories, activity description prediction, and so on. Given a sequence of video frames, one can first use SBGAR to generate captions for these frames. Then, we can use GRIP to generate captions in the future by creating a concept graph over the past generated captions. For more interactive, we can further use a generative model [178] to create frames based on these predicted frames.

As for model compression, it will be interesting to see how DAC performs when being applied to deep learning models in other fields, e.g., voice recognition, language translation, etc. We also want to explore the possibility of adapting DAC on other types of layers, e.g., 3D convolutional layer, compared with other tensor decomposition formats [76, 89]. In addition, DAC is designed to decompose a pre-trained layer into two layers with fewer parameters. Such an approach increases the number of layers of a deep learning model. Considering the fact that in certain application scenarios, e.g., models being run on mobile devices, where it is important to limit the total number of layers a model has, we need to improve DAC so that it can reduce parameters without adding additional layers. Another research direction is to combine low rank constraints with weight decomposition. These constraints could be convex regularizations like nuclear norm and Frobenius norm, or non-convex quasi-norms like Schatten p and TS1 [181, 180, 182]. In addition, some recent work [151, 99] improves the robustness of a deep learning model by using quantization to compress a deep learning model. Their basic intuition is that small perturbations or distortions can

be denoised with low-bit representations. Interested researchers can explore how to integrate DAC and quantization not only to compress deep learning models but also to increase their robustness.

Bibliography

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017.

[3] Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning. In *Advances in neural information processing systems*, pages 577–584, 2003.

[4] Sina Mokhtarzadeh Azar, Mina Ghadimi Atigh, Ahmad Nickabadi, and Alexandre Alahi. Convolutional relational machine for group activity recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7892–7901, 2019.

[5] Mohanad Babiker, Othman O Khalifa, Kyaw Kyaw Htike, Aisha Hassan, and Muhamed Zaharadeen. Automated daily human activity recognition for video surveillance using neural network. In *2017 IEEE 4th International Conference on Smart Instrumentation, Measurement and Application (ICSIMA)*, pages 1–5. IEEE, 2017.

[6] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Sequential deep learning for human action recognition. In *International workshop on human behavior understanding*, pages 29–39. Springer, 2011.

[7] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[8] Hakan Bilen, Basura Fernando, Efstratios Gavves, Andrea Vedaldi, and Stephen Gould. Dynamic image networks for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3034–3042, 2016.

[9] Phil Blunsom, Nando de Freitas, Edward Grefenstette, Karl Moritz Hermann, et al. A deep architecture for semantic parsing. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, 2014.

[10] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. *Computer Vision-ECCV 2004*, pages 25–36, 2004.

[11] Matthias Butenuth, Florian Burkert, Florian Schmidt, Stefan Hinz, Dirk Hartmann, Angelika Kneidl, André Borrmann, and Beril Sirmacek. Integrating pedestrian simulation, tracking and event detection for crowd analysis. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 150–157. IEEE, 2011.

[12] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[13] Francesco Cardile, Giancarlo Iannizzotto, and Francesco La Rosa. A vision-based system for elderly patients monitoring. In *3rd International Conference on Human System Interaction*, pages 195–202. IEEE, 2010.

[14] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[15] Yu-Wei Chao, Jimei Yang, Brian Price, Scott Cohen, and Jia Deng. Forecasting human dynamics from static images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 548–556, 2017.

[16] Bin Chen, Jie Shen, and Helei Sun. A fast face recognition system on mobile phone. In *2012 International Conference on Systems and Informatics (ICSAI2012)*, pages 1783–1786. IEEE, 2012.

[17] Duan-Yu Chen and Po-Chung Huang. Visual-based human crowds behavior analysis based on graph modeling and matching. *IEEE Sensors Journal*, 13(6):2129–2138, 2013.

[18] Enqing Chen, Xue Bai, Lei Gao, Haron Chweya Tinega, and Yingqiang Ding. A spatiotemporal heterogeneous two-stream network for action recognition. *IEEE Access*, 7:57267–57275, 2019.

[19] Y. F. Chen, M. Liu, and J. How. Augmented dictionary learning for motion prediction. *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[20] Kwang-Ting Cheng and Yi-Chu Wang. Using mobile gpu for general-purpose computing–a case study of face recognition on smartphones. In *Proceedings of 2011 International Symposium on VLSI Design, Automation and Test*, pages 1–4. IEEE, 2011.

[21] Hsu-kuang Chiu, Ehsan Adeli, Borui Wang, De-An Huang, and Juan Carlos Niebles. Action-agnostic human pose forecasting. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1423–1432. IEEE, 2019.

[22] Wongun Choi and Silvio Savarese. A unified framework for multi-target tracking and collective activity recognition. In *European Conference on Computer Vision*, pages 215–230. Springer, 2012.

[23] Wongun Choi, Khuram Shahid, and Silvio Savarese. What are they doing?: Collective activity classification using spatio-temporal relationship among people. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1282–1289. IEEE, 2009.

[24] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[25] Chien-Li Chou, Hua-Tsung Chen, and Suh-Yin Lee. Multimodal video-to-near-scene annotation. *IEEE Transactions on Multimedia*, 19(2):354–366, 2016.

[26] James Colyar and John Halkias. Us highway 101 dataset. *Federal Highway Administration (FHWA), Tech. Rep. FHWA-HRT-07-030*, 2007.

[27] James Colyar and John Halkias. Us highway 80 dataset. *Federal Highway Administration (FHWA), Tech. Rep. FHWA-HRT-07-030*, 2007.

[28] Yang Cong, Haifeng Gong, Song-Chun Zhu, and Yandong Tang. Flow mosaicking: Real-time pedestrian counting without scene-specific learning. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1093–1100. IEEE, 2009.

[29] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[30] Hannah M Dee and Alice Caplier. Crowd behaviour analysis using histograms of motion direction. In *2010 IEEE International Conference on Image Processing*, pages 1545–1548. IEEE, 2010.

[31] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[32] Weijian Deng, Liang Zheng, Qixiang Ye, Guoliang Kang, Yi Yang, and Jianbin Jiao. Image-image domain adaptation with preserved self-similarity and domain-dissimilarity for person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 994–1003, 2018.

[33] Zhiwei Deng, Mengyao Zhai, Lei Chen, Yuhao Liu, Srikanth Muralidharan, Mehrsan Javan Roshtkhari, and Greg Mori. Deep structured models for group activity recognition. *arXiv preprint arXiv:1506.04191*, 2015.

[34] Nachiket Deo, Akshay Rangesh, and Mohan M Trivedi. How would surround vehicles move? a unified framework for maneuver classification and motion prediction. *IEEE Transactions on Intelligent Vehicles*, 3(2):129–140, 2018.

[35] Nachiket Deo and Mohan M Trivedi. Convolutional social pooling for vehicle trajectory prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1468–1476, 2018.

[36] Nachiket Deo and Mohan M Trivedi. Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1179–1184. IEEE, 2018.

[37] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[38] Li Du, Zixuan Wang, Leiquan Wang, Zhicheng Zhao, Fei Su, Bojin Zhuang, and Nikolaos V Boulgouris. Adaptive visual interaction based multi-target future state prediction for autonomous driving vehicles. *IEEE Transactions on Vehicular Technology*, 68(5):4249–4261, 2019.

[39] Jiajun Duan, Hao Xu, and Wenxin Liu. Q-learning-based damping control of wide-area power systems under cyber uncertainties. *IEEE Transactions on Smart Grid*, 9(6):6408–6418, 2017.

[40] Jiajun Duan, Zhehan Yi, Di Shi, Chang Lin, Xiao Lu, and Zhiwei Wang. Reinforcement-learning-based optimal control for hybrid energy storage systems in hybrid ac/dc microgrids. *IEEE Transactions on Industrial Informatics*, 2019.

[41] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[42] Hehe Fan, Liang Zheng, Chenggang Yan, and Yi Yang. Unsupervised person re-identification: Clustering and fine-tuning. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 14(4):83, 2018.

[43] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003.

[44] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1933–1941, 2016.

[45] Bin Feng, Fangzi He, Xinggang Wang, Yongjiang Wu, Hao Wang, Sihua Yi, and Wenyu Liu. Depth-projection-map-based bag of contour fragments for robust hand gesture recognition. *IEEE Transactions on Human-Machine Systems*, 47(4):511–523, 2016.

[46] Sarah Ferguson, Brandon Luders, Robert C Grande, and Jonathan P How. Real-time predictive modeling and robust avoidance of pedestrians with uncertain, changing intentions. In *Algorithmic Foundations of Robotics XI*, pages 161–177. Springer, 2015.

[47] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Descriptor matching with convolutional neural networks: a comparison to sift. *arXiv preprint arXiv:1405.5769*, 2014.

[48] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[49] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4346–4354, 2015.

[50] G Garcia-Bunster, M Torres-Torriti, and C Oberli. Crowded pedestrian counting at bus stops from perspective transformations of foreground areas. *IET computer vision*, 6(4):296–305, 2012.

[51] Weina Ge, Robert T Collins, and R Barry Ruback. Vision-based analysis of small groups in pedestrian crowds. *IEEE transactions on pattern analysis and machine intelligence*, 34(5):1003–1016, 2012.

[52] Partha Ghosh, Jie Song, Emre Aksan, and Otmar Hilliges. Learning human motion models for long-term predictions. In *2017 International Conference on 3D Vision (3DV)*, pages 458–466. IEEE, 2017.

[53] Georgia Gkioxari and Jitendra Malik. Finding action tubes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 759–768, 2015.

[54] G. Habibi, N. Jaipuria, and J. P. How. Context aware pedestrian motion prediction in urban intersections. *IEEE International Conference on Robotics & Automation (ICRA)*, 2018.

[55] Hossein Hajimirsadeghi, Wang Yan, Arash Vahdat, and Greg Mori. Visual recognition by counting instances: A multi-instance cardinality potential kernel. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2596–2605, 2015.

[56] Yamin Han, Peng Zhang, Tao Zhuo, Wei Huang, and Yanning Zhang. Going deeper with two-stream convnets for action recognition in video surveillance. *Pattern Recognition Letters*, 107:83–90, 2018.

[57] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[58] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[59] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[60] Rui Hou, Chen Chen, and Mubarak Shah. Tube convolutional neural network (t-cnn) for action detection in videos. *arXiv preprint arXiv:1703.10664*, 2017.

[61] Adam Houenou, Philippe Bonnifait, Véronique Cherfaoui, and Wen Yao. Vehicle trajectory prediction based on motion model and maneuver recognition. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4363–4369. IEEE, 2013.

[62] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[63] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[64] Mostafa S. Ibrahim, Srikanth Muralidharan, Zhiwei Deng, Arash Vahdat, and Greg Mori. A hierarchical deep temporal model for group activity recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[65] Mostafa S. Ibrahim, Srikanth Muralidharan, Zhiwei Deng, Arash Vahdat, and Greg Mori. A hierarchical deep temporal model for group activity recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[66] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *arXiv preprint arXiv:1612.01925*, 2016.

[67] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3. 6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE transactions on pattern analysis and machine intelligence*, 36(7):1325–1339, 2013.

[68] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[69] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016.

[70] Ahmad Jalal, Shaharyar Kamal, and Daijin Kim. A depth video-based human detection and activity recognition using multi-features and embedded hidden markov models for health care monitoring systems. *International Journal of Interactive Multimedia & Artificial Intelligence*, 4(4), 2017.

[71] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

[72] Y.-G. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes. http://crcv.ucf.edu/THUMOS14/, 2014.

[73] Zhixing Jin and Bir Bhanu. Single camera multi-person tracking based on crowd simulation. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3660–3663. IEEE, 2012.

[74] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[75] Taewan Kim, Hyungsoo Park, Sang Hoon Hong, and Yunmo Chung. Integrated system of face recognition and sound localization for a smart door phone. *IEEE Transactions on consumer Electronics*, 59(3):598–603, 2013.

[76] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[77] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[78] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[79] Ron Kohavi and Foster Provost. Glossary of terms. editorial for the special issue on applications of machine learning and the knowledge discovery process. *Machine Learning*, 30(2-3):271–274, 1998.

[80] Raghavendra Kotikalapudi and contributors. keras-vis. https://github.com/raghakot/keras-vis, 2017.

[81] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[82] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 204–211. IEEE, 2017.

[83] Parag Kulkarni, Bhagyashri Patil, and Bela Joglekar. An effective content based video analysis and retrieval using pattern indexing techniques. In *2015 International Conference on Industrial Instrumentation and Control (ICIC)*, pages 87–92. IEEE, 2015.

[84] Iljung S Kwak, David Kriegman, and Kristin Branson. Detecting the starting frame of actions in video. *arXiv preprint arXiv:1906.03340*, 2019.

[85] T. Lan, L. Sigal, and G. Mori. Social roles in hierarchical models for human activity recognition. *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[86] Tian Lan, Yang Wang, and Greg Mori. Discriminative figure-centric models for joint action localization and recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2003–2010. IEEE, 2011.

[87] Tian Lan, Yang Wang, Weilong Yang, Stephen N Robinovitch, and Greg Mori. discriminativeminative latent models for recognizing contextual group activities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(8):1549–1562, 2012.

[88] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, pages 98–113, 1997.

[89] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[90] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[91] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *ROBOMECH journal*, 1(1):1, 2014.

[92] S. Lefvre, D. Vasquez, and C. Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *Robomech Journal*, 2014.

[93] Marius Leordeanu, Andrei Zanfir, and Cristian Sminchisescu. Locally affine sparse-to-dense matching for motion and occlusion estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1721–1728, 2013.

[94] Xin Li and Mooi Choo Chuah. Sbgar: Semantics based group activity recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2876–2885, 2017.

[95] Xin Li and Mooi Choo Chuah. Rehar: Robust and efficient human activity recognition. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 362–371. IEEE, 2018.

[96] Xin Li, Xiaowen Ying, and Mooi Choo Chuah. Grip: Graph-based interaction-aware trajectory prediction. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019.

[97] Xin Li, Shuai Zhang, Bolan Jiang, Yingyong Qi, Mooi Choo Chuah, and Ning Bi. Dac: Data-free automatic acceleration of convolutional networks. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1598–1606. IEEE, 2019.

[98] Xiaodan Liang, Lisa Lee, Wei Dai, and Eric P Xing. Dual motion gan for future-flow embedded video prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1744–1752, 2017.

[99] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. In *International Conference on Learning Representations (ICLR)*, 2019.

[100] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[101] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[102] Yutian Lin, Liang Zheng, Zhedong Zheng, Yu Wu, Zhilan Hu, Chenggang Yan, and Yi Yang. Improving person re-identification by attribute and identity learning. *Pattern Recognition*, 2019.

[103] Shiwen Liu, Kan Zheng, Long Zhao, and Pingzhi Fan. A driving intention prediction method based on hidden markov model for autonomous driving. *arXiv preprint arXiv:1902.09068*, 2019.

[104] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[105] Xi Liu, Lei Liu, Steven J Simske, and Jerry Liu. Human daily activity recognition for healthcare using wearable and visual sensing data. In *2016 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 24–31. IEEE, 2016.

[106] Ziwei Liu, Raymond A Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video frame synthesis using deep voxel flow. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4463–4471, 2017.

[107] William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

[108] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[109] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[110] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3569–3577, 2018.

[111] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[112] Yuexin Ma, Xinge Zhu, Sibo Zhang, Ruigang Yang, Wenping Wang, and Dinesh Manocha. Trafficpredict: Trajectory prediction for heterogeneous traffic-agents. *arXiv preprint arXiv:1811.02146*, 2018.

[113] Julieta Martinez, Michael J Black, and Javier Romero. On human motion prediction using recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2891–2900, 2017.

[114] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[115] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[116] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[117] Jose Hernando Mosquera, Humberto Loaiza, Sandra Esperaza Nope, and Andres David Restrepo. Identifying facial gestures to emulate a mouse: navigation application on facebook. *IEEE Latin America Transactions*, 15(1):121–128, 2017.

[118] Chee Kiat Ng, Marios Savvides, and Pradeep K Khosla. Real-time face verification system on a cell-phone using advanced correlation filters. In *Fourth IEEE Workshop on Automatic Identification Advanced Technologies (AutoID'05)*, pages 57–62. IEEE, 2005.

[119] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015.

[120] Yu Pan, Jing Xu, Maolin Wang, Jinmian Ye, Fei Wang, Kun Bai, and Zenglin Xu. Compressing recurrent neural networks with tensor ring for action recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4683–4690, 2019.

[121] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[122] Keyurkumar Patel, Hu Han, and Anil K Jain. Secure face unlock: Spoof detection on smartphones. *IEEE transactions on information forensics and security*, 11(10):2268–2283, 2016.

[123] Viorica Patraucean, Ankur Handa, and Roberto Cipolla. Spatio-temporal video autoencoder with differentiable memory. *arXiv preprint arXiv:1511.06309*, 2015.

[124] Dario Pavllo, David Grangier, and Michael Auli. Quaternet: A quaternion-based recurrent model for human motion. *arXiv preprint arXiv:1805.06485*, 2018.

[125] Xiaojiang Peng and Cordelia Schmid. Multi-region two-stream r-cnn for action detection. In *European Conference on Computer Vision*, pages 744–759. Springer, 2016.

[126] Andrea Prati, Caifeng Shan, and Kevin I-Kai Wang. Sensors, vision and networks: From video surveillance to activity recognition and health monitoring. *Journal of Ambient Intelligence and Smart Environments*, 11(1):5–22, 2019.

[127] V. Ramananthan, B. Yao, and F. F. Li. Social role discovery in human events. *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[128] Vignesh Ramanathan, Jonathan Huang, Sami Abu-El-Haija, Alexander Gorban, Kevin Murphy, and Li Fei-Fei. Detecting events and key actors in multi-person videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3043–3053, 2016.

[129] Vignesh Ramanathan, Jonathan Huang, Sami Abu-El-Haija, Alexander Gorban, Kevin Murphy, and Li Fei-Fei. Detecting events and key actors in multi-person videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3043–3053, 2016.

[130] Carl E Rasmussen and Zoubin Ghahramani. Infinite mixtures of gaussian process experts. In *Advances in neural information processing systems*, pages 881–888, 2002.

[131] Siddharth S Rautaray and Anupam Agrawal. Real time hand gesture recognition system for dynamic applications. *International Journal of UbiComp*, 3(1):21, 2012.

[132] Mikel D Rodriguez, Javed Ahmed, and Mubarak Shah. Action mach a spatio-temporal maximum average correlation height filter for action recognition. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[133] Dan Rosenbaum, Daniel Zoran, and Yair Weiss. Learning the local statistics of optical flow. In *Advances in Neural Information Processing Systems*, pages 2373–2381, 2013.

[134] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

[135] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[136] Julian Schlechtriemen, Florian Wirthmueller, Andreas Wedel, Gabi Breuel, and Klaus-Dieter Kuhnert. When will it change the lane? a probabilistic regression approach for rarely occurring events. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1373–1379. IEEE, 2015.

[137] Matthias Schreier, Volker Willert, and Jürgen Adamy. Bayesian, maneuver-based, long-term trajectory prediction and criticality assessment for driver assistance systems. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 334–341. IEEE, 2014.

[138] Nima Sedaghat. Next-flow: Hybrid multi-tasking with next-frame prediction to boost optical-flow estimation in the wild. *arXiv preprint arXiv:1612.03777*, 1(2):6, 2016.

[139] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *See https://arxiv. org/abs/1610.02391 v3*, 2016.

[140] Yiran Shen, Wen Hu, Mingrui Yang, Bo Wei, Simon Lucey, and Chun Tung Chou. Face recognition on smartphones via optimised sparse representation classification. In *Proceedings of the 13th international symposium on Information processing in sensor networks*, pages 237–248. IEEE Press, 2014.

[141] Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, and Mickey Aleksic. A quantization-friendly separable convolution for mobilenets. *arXiv preprint arXiv:1803.08607*, 2018.

[142] Jianbo Shi and Carlo Tomasi. Good features to track. Technical report, Cornell University, 1993.

[143] Huang-Chia Shih. A survey of content-aware video analysis for sports. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(5):1212–1231, 2017.

[144] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[145] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[146] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[147] K. Soomro, A. Roshan Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. In *CRCV-TR-12-01*, 2012.

[148] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[149] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

[150] Manuel Stein, Halldór Janetzko, Andreas Lamprecht, Daniel Seebacher, Tobias Schreck, Daniel Keim, and Michael Grossniklaus. From game events to team tactics: Visual analysis of dangerous situations in multi-match data. In *2016 1st International Conference on Technology and Innovation in Sports, Health and Wellbeing (TISHW)*, pages 1–9. IEEE, 2016.

[151] Zhun Sun, Mete Ozay, Yan Zhang, Xing Liu, and Takayuki Okatani. Feature quantization for defending against distortion of images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7957–7966, 2018.

[152] Mikhail Sysoev, Andrej Kos, Jože Guna, and Matevž Pogačnik. Estimation of the driving style based on the usersâĂŹ activity and environment influence. *Sensors*, 17(10):2404, 2017.

[153] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.

[154] Urmish Thakker, Jesse Beu, Dibakar Gope, Ganesh Dasika, and Matthew Mattina. Run-time efficient rnn compression for inference on edge devices. In *Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications at International Symposium of Computer Architecture*, 2019.

[155] Rafael Toledo-Moreo and Miguel A Zamora-Izquierdo. Imm-based lane-change prediction in highways with low-cost gps/ins. *IEEE Transactions on Intelligent Transportation Systems*, 10(1):180–185, 2009.

[156] Martin Torstensson, Thanh Hai Bui, David Lindström, Cristofer Englund, and Boris Duran. In-vehicle driver and passenger activity recognition. 2019.

[157] Du Tran, Lubomir D Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3d: generic features for video analysis. *CoRR, abs/1412.0767*, 2(7):8, 2014.

[158] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

[159] Quan Tran and Jonas Firl. Online maneuver recognition and multimodal trajectory prediction for intersection assistance using non-parametric regression. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 918–923. IEEE, 2014.

[160] Amin Ullah, Khan Muhammad, Ijaz Ul Haq, and Sung Wook Baik. Action recognition using optimized deep autoencoder and cnn for surveillance data streams of non-stationary environments. *Future Generation Computer Systems*, 96:386–397, 2019.

[161] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[162] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE transactions on pattern analysis and machine intelligence*, 2016.

[163] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3169–3176. IEEE, 2011.

[164] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer, 2016.

[165] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8445–8453, 2019.

[166] Andreas Wedel, Daniel Cremers, Thomas Pock, and Horst Bischof. Structure-and motion-adaptive regularization for high accuracy optic flow. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1663–1668. IEEE, 2009.

[167] Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Learning to track for spatio-temporal action localization. In *Proceedings of the IEEE international conference on computer vision*, pages 3164–3172, 2015.

[168] Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[169] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 305–321, 2018.

[170] Yang Xing, Chen Lv, Huaji Wang, Dongpu Cao, Efstathios Velenis, and Fei-Yue Wang. Driver activity recognition for intelligent vehicles: A deep learning approach. *IEEE Transactions on Vehicular Technology*, 2019.

[171] Guogang Xiong, Xinyu Wu, Yen-lun Chen, and Yongsheng Ou. Abnormal crowd behavior detection based on the energy model. In *2011 IEEE International Conference on Information and Automation*, pages 495–500. IEEE, 2011.

[172] Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. Deep neural network compression with single and multiple level quantization. *arXiv preprint arXiv:1803.03289*, 2018.

[173] Tianfan Xue, Jiajun Wu, Katherine Bouman, and Bill Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Advances in neural information processing systems*, pages 91–99, 2016.

[174] Song Yao, Song Han, Kaiyuan Guo, Jianqiao Wangni, and Yu Wang. Hardware-friendly convolutional neural network with even-number filter size. 2016.

[175] Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. Binaryrelax: A relaxation approach for training deep neural networks with quantized weights. *arXiv preprint arXiv:1801.06313*, 2018.

[176] Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. Blended coarse gradient descent for full quantization of deep neural networks. *arXiv preprint arXiv:1808.05240*, 2018.

[177] Penghang Yin, Shuai Zhang, Yingyong Qi, and Jack Xin. Quantization and training of low bit-width convolutional neural networks for object detection. *Journal of Computational Mathematics*, 2019.

[178] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5907–5915, 2017.

[179] Shanghang Zhang, Guanhang Wu, Joao P Costeira, and José MF Moura. Fcn-rlstm: Deep spatio-temporal neural networks for vehicle counting in city cameras. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3687–3696. IEEE, 2017.

[180] Shuai Zhang and Jack Xin. Minimization of transformed l_1 penalty: Closed form representation and iterative thresholding algorithms. *Communications in Mathematical Sciences*, 2017.

[181] Shuai Zhang and Jack Xin. Minimization of transformed l_1 penalty: theory, difference of convex function algorithm, and robust application in compressed sensing. *Mathematical Programming*, 2018.

[182] Shuai Zhang, Penghang Yin, and Jack Xin. Transformed schatten-1 iterative thresholding algorithms for low rank matrix completion. *Communications in Mathematical Sciences*, 2017.

[183] Weiyu Zhang, Menglong Zhu, and Konstantinos G Derpanis. From actemes to action: A strongly-supervised representation for detailed action understanding. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2248–2255, 2013.

[184] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices, 2017.

[185] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2016.

[186] Jiaojiao Zhao and Cees GM Snoek. Dance with flow: Two-in-one stream action detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9935–9944, 2019.

[187] Feng Zhu, Xiaogang Wang, and Nenghai Yu. Crowd tracking with dynamic evolution of group structures. In *European conference on computer vision*, pages 139–154. Springer, 2014.

[188] Guangming Zhu, Liang Zhang, Peiyi Shen, and Juan Song. Multimodal gesture recognition using 3-d convolution and convolutional lstm. *IEEE Access*, 5:4517–4524, 2017.

[189] Guangyu Zhu, Changsheng Xu, Qingming Huang, Yong Rui, Shuqiang Jiang, Wen Gao, and Hongxun Yao. Event tactic analysis based on broadcast sports video. *IEEE Transactions on Multimedia*, 11(1):49–67, 2008.

[190] Yi Zhu, Zhenzhong Lan, Shawn Newsam, and Alexander Hauptmann. Hidden two-stream convolutional networks for action recognition. In *Asian Conference on Computer Vision*, pages 363–378. Springer, 2018.

Curriculum Vita

Xin Li

EDUCATION

Lehigh University, Bethlehem, PA, USA 09/2015 - 08/2019
Ph.D. in Computer Science

Peking University, Beijing, China 09/2012 - 08/2015
M.S. in Software Engineering

Qingdao Technological University, Qingdao, Shandong, China 09/2008 - 08/2012
B.E in Software Engineering

WORKING EXPERIENCE

WINS Lab, Lehigh University, Bethlehem, PA, USA 06/2015 - 05/2019
Research Assistant

Samsung Research America Inc. (SRA), Mountain View, CA, USA 05/2019 - 08/2019
Research Intern

Qualcomm Technologies, Inc., San Diego, CA, USA 05/2018 - 08/2018
Interim Engineering Intern

Lehigh University, Bethlehem, PA, USA 01/2018 - 05/2018
Teaching Assistant (CSE327 Artificial Intelligence Theory and Practice)

Qualcomm Technologies, Inc., San Diego, CA, USA 05/2017 - 08/2017
Interim Engineering Intern

Lehigh University, Bethlehem, PA, USA 01/2017 - 05/2017
Teaching Assistant (CSE202 Computer Organization and Architecture)

PUBLICATIONS

- **Xin Li**, Xiaowen Ying, Mooi Choo Chuah. GRIP: Graph-based Interaction-aware Trajectory Prediction. In IEEE Intelligent Transportation Systems Conference (ITSC), 2019.

- **Xin Li**, Shuai Zhang, Bolan Jiang, Yingyong Qi, Mooi Choo Chuah, Ning Bi. DAC: Data-free Automatic Acceleration of Convolutional Networks. In IEEE Winter Conference on Applications of Computer Vision (WACV), 2019.

- Xiaowen Ying, **Xin Li**, Mooi Choo Chuah. LiveFace: A Multi-Task CNN for Fast Face-Authentication. In IEEE International Conference on Machine Learning and Applications (ICMLA), 2018.

- **Xin Li**, Mooi Choo Chuah. ReHAR: Robust and Efficient Human Activity Recognition. In IEEE Winter Conference on Applications of Computer Vision (WACV), 2018.

- **Xin Li**, Mooi Choo Chuah. SBGAR: Semantics based group activity recognition. In IEEE Conference on Computer Vision and Pattern Recognition (ICCV), 2017.

- **Xin Li**, Mooi Choo Chuah, Subhrajit Bhattacharya. UAV assisted smart parking solution. In International Conference on Unmanned Aircraft Systems (ICUAS), 2017.

- **Xin Li**, Qinghan Xue, Mooi Choo Chuah. CASHEIRS: Cloud assisted scalable hierarchical encrypted based image retrieval system. In IEEE Conference on Computer Communications (INFOCOM), 2017.